NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

THE DESIGN OF AN INTERFACE EDITOR FOR THE COMPUTER-AIDED PROTOTYPING SYSTEM

by

Bruce D. Plutchak

September 1997

Thesis Advisor: Luqi

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1997	Master's Thesis

4. TITLE AND SUBTITLE

THE DESIGN OF AN INTERFACE EDITOR FOR THE COMPUTER-AIDED PROTOTYPING SYSTEM

5. FUNDING NUMBERS

6. AUTHOR

Plutchak, Bruce, D.

7. PERFORMING ORGANIZATION NAME AND ADDRESS

Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSOR/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense of the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

ABSTRACT (Maximum 200 words)

This thesis focuses on the design and implementation of a new interface editor for the Computer-Aided Prototyping System (CAPS), which de-couples the user interface from the real-time prototype. Using this design, a CAPS user creates a prototype with an interface development tool and a Prototyping System Description Language (PSDL) editor. This real-time prototype executes on two processors using a client/server architecture; the user interface executes on a client, and the real-time PSDL application executes on a server. In addition, this thesis includes demonstrations, with source code, which implement the design. The demonstrations show that Java development tools can be used to create a high-quality user interface for a PSDL application. A socket connection was used to implement the client/server communication. The demonstrations were successful, but the socket programming model is too primitive for the new design. Therefore, a high-level client/server architecture, such as the Common Object Resource Broker Architecture (CORBA), is required for future development of the design.

14. SUBJECT TERMS	15. NUMBER OF PAGES
CAPS, Software Reuse, Software Base, Real-time, Graphical User Interface, Client/Server, Java	132
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	**UNCLASSIFIED**	**UNCLASSIFIED**	**UL**

Approved for public release; distribution is unlimited

THE DESIGN OF AN INTERFACE EDITOR FOR THE COMPUTER-AIDED PROTOTYPING SYSTEM

Bruce D. Plutchak
B.S., University of California at San Diego, 1984

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SOFTWARE ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1997

ABSTRACT

This thesis focuses on the design and implementation of a new interface editor for the Computer-Aided Prototyping System (CAPS), which de-couples the user interface from the real-time prototype. Using this design, a CAPS user creates a prototype with an interface development tool and a Prototyping System Description Language (PSDL) editor. This real-time prototype executes on two processors using a client/server architecture; the user interface executes on a client, and the real-time PSDL application executes on a server. In addition, this thesis includes demonstrations, with source code, which implement the design. The demonstrations show that Java development tools can be used to create a high-quality user interface for a PSDL application. A socket connection was used to implement the client/server communication. The demonstrations were successful, but the socket programming model is too primitive for the new design. Therefore, a high-level client/server architecture, such as the Common Object Resource Broker Architecture (CORBA), is required for future development of the design.

THESIS DISCLAIMER

Ada is a trademark of the United States Government, Ada Joint Program Office.
Motif is a trademark of Open Software Foundation.
X-Windows is a trademark of The Massachusetts Institute of Technology.
Java is a trademark of Sun Microsystems.
All other trademarks and registered trademarks herein are the property of their owners.

TABLE OF CONTENTS

ACKNOWLEDGMENT

The author wants to thank Prof. Luqi and Prof. Berzins for their help and guidance in preparing this work. I would also like my thank Lori, Diana, and Kyle for their support.

I. INTRODUCTION

A. GENERAL

In the development of software applications, the user interface software is often complex, large, and difficult to design. The user interface is the aspect of a computer program that has the most direct contact with the user. Likewise, the design of user interfaces has become an important part of the Computer-Aided Prototyping System (CAPS). CAPS is a collection of engineering tools, which are used to design real-time systems. A CAPS user enters specifications using the Prototyping System Description Language (PSDL). To complete the prototype, the user designs a user interface with the CAPS interface editor. Finally, the PSDL is translated into Ada; the Ada is linked with the user interface; and the prototype application is executed.

CAPS is an evolving research project, and many of its components are still under development. The user interface development tools of CAPS need to be revised. For instance, the current version of CAPS is only fully functional on a Sun Microsystems workstation running SunOS UNIX. Currently, the CAPS interface editor is Century Computing's Transportable Applications Environment (TAE). TAE is only available for a few platforms, and it has limited capabilities. In addition, the CAPS user interface, written in Motif, is not easily ported to other platforms.

B. PROBLEM STATEMENT

A CAPS designer uses the prototyping process to create an application. PSDL is used to express the specifications of the application. Next, the CAPS interface editor can be used to create a prototype user interface. Unfortunately, the current interface editor does have the support and functionality that is required. This thesis focuses on creating a new design for linking a user interface with a PSDL application. In addition, this thesis

evaluates user interface development tools, which could be used to increase the functionality, portability, and versatility of CAPS.

C. SCOPE

The scope of this work was originally intended to integrate a commercial user interface development tool into CAPS. As work on this thesis progressed, it became clear that a new design for linking the user interface with the prototype was required. This design is based upon a client/server architecture. In addition, two demonstrations that use the new design are included.

D. ORGANIZATION OF THESIS

Chapter II contains background information about CAPS, user interface development tools, and supporting documentation. Chapter III presents the design process and results. Chapter IV presents the implementation of the new design. Chapter V contains demonstrations of the new design. Chapter VI provides the conclusion and a discussion of future research. Appendix A contains evaluations of user interface development tools.

II. BACKGROUND

A. COMPUTER-AIDED PROTOTYPING SYSTEM

The Computer-Aided Prototyping System (CAPS) is a software engineering tool for developing prototypes of real-time systems [Ref. 1]. CAPS was designed to allow a user to build software specifications and make them executable. In addition, CAPS can be used for requirements analysis, evaluation of models, and designing large-embedded systems. CAPS promotes the rapid prototyping life cycle (Figure 1). Rapid prototyping is an alternative paradigm for software development. By using the prototyping process, a user can validate system requirements early in the project's life cycle. In addition, CAPS includes tools for software reuse and evolution. [Ref. 2]

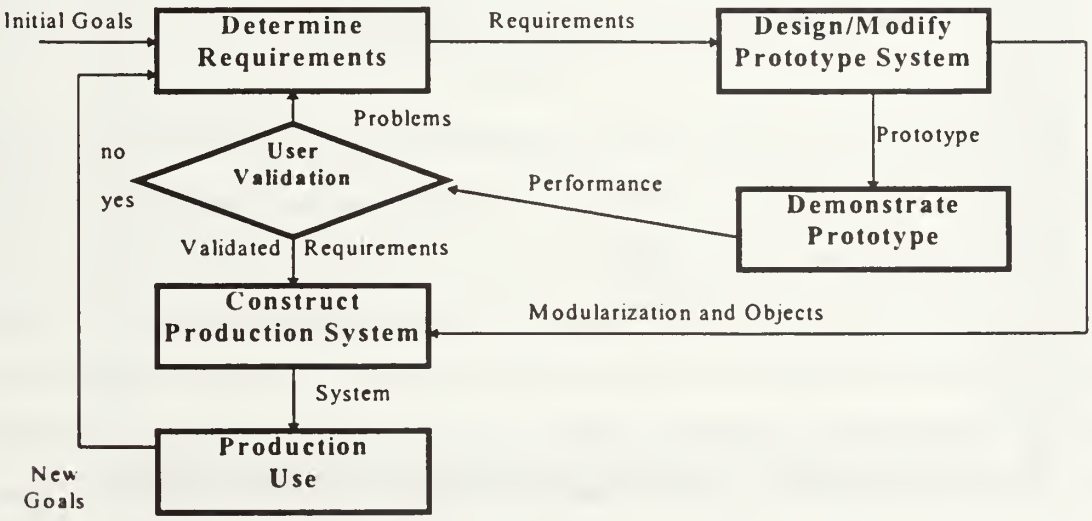

Figure 1. The Rapid Prototyping Life Cycle "From Ref. [7]"

A CAPS user specifies relationships, data flow, and constraints between software objects. These specifications are expressed in the Prototyping System Description Language (PSDL). The user initially works at a high level of abstraction, breaking down the objects into lower levels of abstraction. At the lowest levels, existing

code from a software library is used to implement the object. Alternatively, a high-level language, such as Ada, can be used to implement the object. The papers [Ref. 3], [Ref. 4], [Ref. 5], [Ref. 6], and [Ref. 7] provide more information about CAPS.

1. CAPS Components

CAPS is a development environment consisting of tools linked by a user interface (Figure 2). The main sections of CAPS include: *editors, execution support, software database,* and *project control.* The *editors* are used to implement the software design. *Execution support* is for software translation, real-time scheduling, and compilation. The *software database* is intended for software storage and reuse. *Project control* contains tools for software evolution, such as revision control, merging of software versions, and project planning. [Ref. 8]

2. Prototyping System Description Language

A CAPS user can input software specifications via the Prototyping System Description Language (PSDL). PSDL is a text-based language designed to express specifications of real-time systems. PSDL has only two kinds of components: *operators* and *types*. PSDL is based on a graph model of edges and vertices (Figure 3). The edges represent *streams* of data flow from one operator to another. The streams are instances of types. The vertices represent software *operators*. The operators can either be *sporadic* or *periodic*. The operators may have associated timing constraints. An example timing constraint is Maximum Execution Time (MET). In PSDL, the MET of an operator must not be exceeded, otherwise a timing error will be displayed. In addition, the operators may also have control constraints. Control constraints for a periodic operator include: Finish Within (FW) and Period. [Ref. 3]

PSDL supports the concept of program abstraction. The PSDL operators can either be *composite* or *atomic*. An atomic operator is implemented in a standard programming

language such as Ada. A composite operator can be decomposed into a sub-layer containing operators and streams. [Ref. 3]

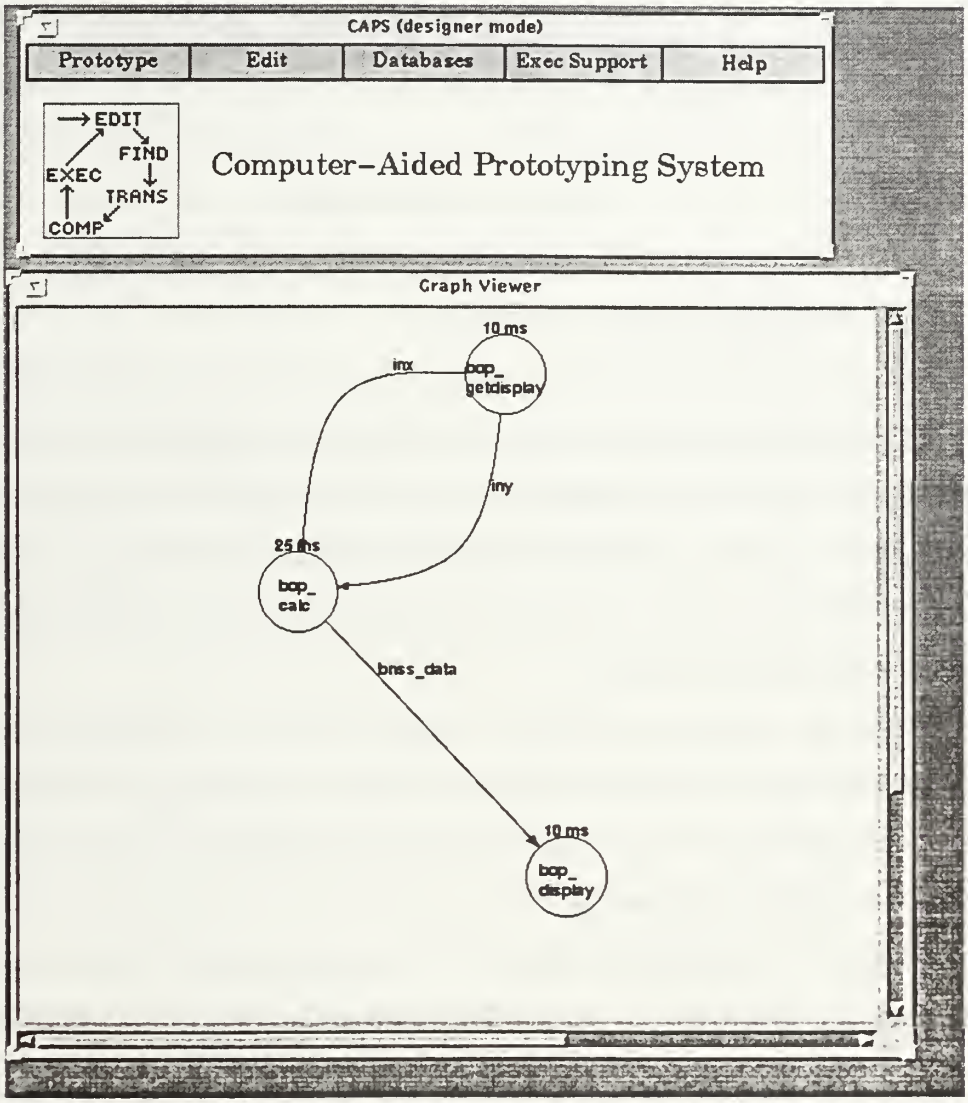

Figure 2. CAPS and Graph Editor

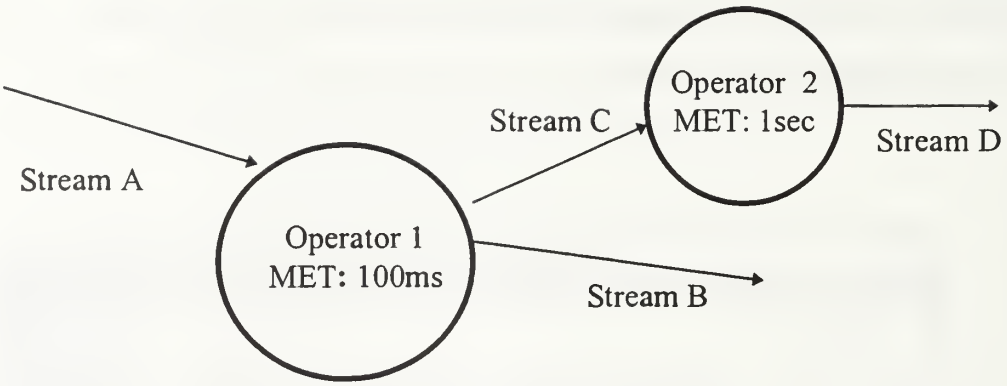

Figure 3. PSDL Graph

3. Monitoring PSDL Execution

By using PSDL, a CAPS user can create a prototype application. In most cases, a CAPS user must be able to monitor the prototype's execution. Even if the prototype is destined to be an embedded application without a Graphical User Interface (GUI), the user may require a GUI to evaluate the prototype. Alternatively, a user may also require a GUI to emulate an external environment.

CAPS has built-in diagnostics, which monitor a prototype's execution. Additionally, CAPS monitors the execution times of the operators. The user will be notified if timing constraints are violated. CAPS prints data-stream errors and exceptions during the execution of the prototype.

In addition to diagnostic output, a CAPS user may wish to observe data values, provide input, and observe simulations. A user can add simple print statements to monitor execution of a PSDL application. This is traditionally done by adding Ada print statements to the atomic operator's implementation.

4. Interface Editor

The interface editor is a GUI development tool designed to help the developer create high-quality user interfaces in a short period of time. If a CAPS user requires more

than just simple print statements in the prototype application, he/she can use the CAPS interface editor to design a graphical user interface prototype (*GUI-prototype*). The GUI-prototype allows a user to display and input information in a window-based environment. In many real-time embedded applications, the GUI-prototype is only required for analysis. In most cases, the GUI-prototype does not have timing constraints. It is important that the GUI-prototype not interfere with the real-time execution of the prototype application. [Ref. 8][Ref. 9]

5. CAPS Release 1.1 Interface Editor: TAE

CAPS Version 1.1 currently incorporates Century Computing's TAE as the interface editor [Ref. 8][Ref. 10]. TAE allows a user to quickly create X-Widows/Motif based GUI-prototypes. TAE generates C or Ada program code. To create a prototype application, the TAE-Ada code is linked with translated PSDL.

TAE has some very useful features, including a GUI builder. In addition, the TAE resource file allows widgets to be interchanged without recompiling the entire application. Also, TAE allows the building of high-level reusable components. TAE has been successfully integrated, documented, tested, and demonstrated with the CAPS system.

However, TAE has drawbacks with respect to CAPS. TAE is expensive to support, license, and to include in the release of CAPS. In addition, TAE is not platform independent. Also, TAE does not work with all versions of UNIX. Most importantly, the current integration of TAE alters the TAE event manager. TAE is linked with a prototype application by removing the infinite loop from the TAE event manager. As a result, the prototype application is responsible for polling the TAE widgets. This design can make the prototype application sluggish to user input and slow when displaying data.

For information on how to link in TAE and PSDL consult [Ref. 8]. For information on how to use TAE, consult [Ref. 10].

B. GUI DEVELOPMENT TOOLS

Graphical User Interface (GUI) design is a field in computer science that has been changing rapidly in the last ten years. Window-based operating systems have allowed applications to have sophisticated user interfaces. Unfortunately, user interface software is often complex, large, and difficult to program. Traditionally, programmers have used toolkits to develop GUIs. The toolkits use the functionality of the platform's windowing and operating system. Fortunately, new software development tools are now available to aid the software designer. A GUI development tool is defined as *software* to aid in the creation of graphical user interfaces. Most GUI development tools are based on toolkits to achieve their look-and-feel. A few development tools do not make direct calls to the standard toolkits, but emulate the look-and-feel of the toolkits. Figure 4 shows the layering of GUI software.

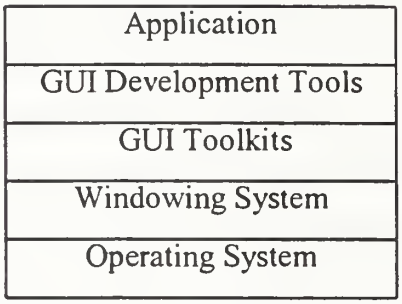

Figure 4. Components of GUI Software "After Ref. [11]"

The software designer must decide whether to use a GUI development tool or program manually with just the toolkit libraries. The decision to program with GUI development tools is better than manual coding in several ways. For instance, most tools support prototyping, reuse, and code consistency. In contrast, the drawbacks of tools could include: added complexity, a higher learning curve, added expense, and performance penalties. [Ref. 12]

Object oriented design and coding has become a fundamental part of most GUI development tools. Most GUI development tools incorporate abstraction and

encapsulation in their design. For example, a software developer may not care to understand the inner functionality of a widget, but he/she may just need to know the available methods for the widget.

The GUI development tools can be grouped by the languages supported. Alternatively, the tools can also be grouped by the platforms supported. The following paragraphs divide the tools based on functionality. However, many of the commercial tools do not fit into just one category. [Ref. 12]

1. GUI Toolkits

A GUI toolkit is a library of widgets that can be used to develop user interfaces. Typical widgets include: buttons, sliders, text-fields, and canvases. Some toolkits contain high-level widgets, such as pop-up dialog boxes, which aid the developer in creating common interfaces. The use of high-level widgets maintains a consistent look-and-feel across different applications. Motif and OpenLook are examples of X-Windows toolkits. An example of a platform independent toolkit is Artificial Intelligence Applications Institute's wxWindows [Ref. 13]. It is a free-of-charge library for C++.

2. GUI Builders

Usually a GUI development product contains a GUI builder to construct and edit the interface. Most GUI builders allow the designer to quickly create an interface in a *What You See Is What You Get* (WYSIWYG) mode. A GUI builder allows the user to easily add, delete, and replace widgets. In many cases, a GUI builder can be used by a person with little programming experience. In the prototyping process, the ability to create GUIs in a short period of time is very important. The GUI builders available today vary in quality, ease of use, and training required.

3. User Interface Management Systems

A User Interface Management System (UIMS) adds functionality to the GUI, without the user having to program in a high-level language. To achieve this functionality, some UIMS include high-level scripting languages, which add behavior to the GUI. [Ref. 14]. Open Software Associates' OpenUI [Ref. 15] and Aonix's Teleuse [Ref. 16] are examples of UIMSs.

4. Analysis, Plotting, Graphics, and Visualization

Some GUI tools contain graphics and 2D-3D visualization tools. Other tools may allow the user to analyze and plot technical data in generic formats. Visual Numerics PV-WAVE [Ref. 17] and Mathworks' Matlab [Ref. 18] are examples. Some tools support real-time analysis of data. DataViews Corporation's DataViews is an example of this kind of tool [Ref. 19].

5. Application Development Environments

Application development environments contain a wide range of tools, in addition to the GUI tools. These products contain tools to aid in the full range of software development. These tools could include: group-ware, code-analysis, debugging, evolution control, and real-time control. Visix's Galaxy [Ref. 20] and Sun Microsystem's Visual Workshop [Ref. 21] are examples of application development environments.

6. Platform Independent Graphical User Interface

A Platform Independent Graphical User Interface (PIGUI) is defined as a software library that supports at least two different operating systems. [Ref. 22] A PIGUI allows a developer to create one version of code that runs on multiple platforms. A PIGUI will

hopefully decrease the amount of total development time for a project supporting multiple platforms. The PIGUI can either preserve the native look-and-feel of the target platform or maintain a consistent look-and-feel across platforms.

There are two types of PIGUIs. The PIGUI can link to the windowing toolkit on the target machine. XVT Softwares's XVT is an example of this kind of PIGUI. [Ref. 23] Secondly, a PIGUI can re-implement the widgets for each target platform. Visix's Galaxy is an example of this type. [Ref. 20] The extra overhead of PIGUIs will generally slow down the execution of the application. The supported languages for PIGUIs include C, C++, Ada, Java, and others, but the most predominate is C++. [Ref. 22]

C. DE-COUPLING OF THE GUI

GUI complexity has increased rapidly in the last few years. Users are requiring well-planned and sophisticated GUIs. The computer applications of the 1960's and 1970's had simple user interfaces with very little operator involvement. These early computer programs were mostly one-tier applications. A *tier* is a component of an application that is bound through external interfaces to other modules of the application [Ref. 24]. One-tier applications include all the functionality in one program. This functionality can include: *presentation management*, *program rules code*, and *database access* (Figure 5). One-tier applications have the advantage of being easy to design, setup, and maintain. On the other hand, one-tier applications are not easily scaleable and have inadequate performance at high volume. [Ref. 25]

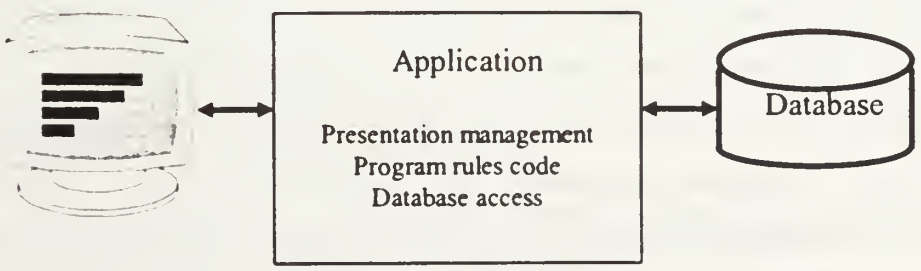

Figure 5. One Tier Application

The recent development of client/server middleware and Database Management Systems (DBMSs) allowed the developer to separate the database code from the rest of the application. Usually, client/server computing involves two or more computers distributing tasks appropriate to each computer to complete the application. Client/Server computing has been widely used in database systems. Figure 6 shows a DMBS task communicating with a database. The DBMS is de-coupled from the rest of the application. [Ref. 15]

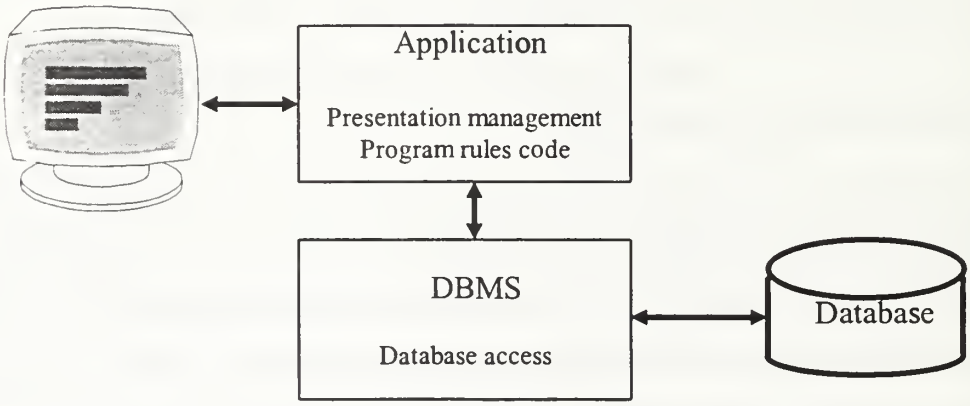

Figure 6. Two-tier Client/Server Application

The separation of the GUI from the application code has mirrored the de-coupling of DBMSs [Ref. 15]. In recent years, the GUI has become the most complex part of many applications. A 1992 study found that an average of 48% of an application code is devoted to the user interface, and about 50% of the implementation time is devoted to the user interface [Ref. 12]. In many software designs the GUI is de-coupled from the rest of the application. Figure 7 shows a three-tier application; the GUI controls the display, handles user input, and requests services from other tiers of the application.

Multiple-tiered computing has also distributed the tasks of an application to many remote sites. Distributed applications use client/server protocols to allow the client application to communicate with a server computer system. Distributed computing makes the sharing of data and processor resources possible.

Client/server computing has advantages and disadvantages. By separating an application into multiple tiers and incorporating the client/server model, it is modularized; reuse is promoted; and the computing load is distributed [Ref. 24]. Even if a client/server application is run on one CPU, it retains the benefits of modularity and reuse. On the other hand, a disadvantage of client/server computing is communication complexity. The communication protocols can add CPU load to the design. Also, client/server systems are harder to setup and design. [Ref. 26]

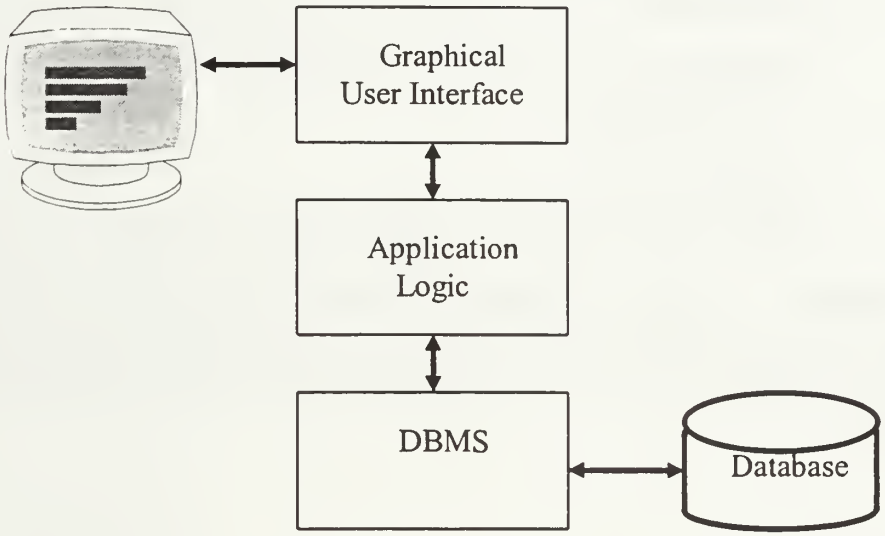

Figure 7. Three-Tier Software Design: Separation of GUI and DBMS

D. EVALUATION OF GUI TOOLS

The number of commercial and public GUI development tools is large and is growing rapidly. The GUI tools vary in complexity, price, quality, programming language support, platform support, and adherence to standards. The following issues were considered in the evaluation of GUI development tools:

- Type of Tool: toolkit, GUI builder, UIMS, development environment, ...
- Toolkits supported: Openlook, Motif, Java, ...
- Platform independence
- Platform support

- Operating system support
- Cost: initial purchase, maintenance
- Level of vendor support
- Future support of tools
- Language Support - Ada, C, C++, Java, ...
- GUI builder availability
- Compliance to standards
- CAPS integration complexity
- Ease of use, learning curve
- Ease of installation
- Documentation
- Complexity
- Prototyping features
- Visualization tools
- Plotting and graphics
- Additional tools: maketool, evolution control, debugging, group-ware, ...

Appendix A contains evaluations of GUI development tools and toolkits. Additional information about these and other tools can be found in the papers [Ref. 22] and [Ref. 25].

III. DESIGN OF A CAPS INTERFACE EDITOR

A. DESIGN PROCESS

As mentioned in the introduction, this thesis focuses on the development of a new interface editor for CAPS. The first step is determining the purpose of the work. The next step is to evaluate the current CAPS' programming environment and determine the assumptions, goals, and constraints for a new design. Since CAPS is a group effort, this thesis must maintain consistency with the previously developed applications.

1. Purpose

The purpose of the CAPS interface editor is to allow a CAPS user to create a GUI-prototype for a prototype application. In addition, the GUI development tools could be used to port the CAPS tools, such as Graph Editor (GE), to other platforms.

2. Assumptions

1. The CAPS user will be familiar with the capabilities of the hardware/software platform.
2. The CAPS user will have knowledge of computer window systems.
3. The CAPS user will be familiar with CAPS, PSDL, and the chosen programming language.

3. Goals/Constraints

1. Goals for an Interface Editor

    1.1 The interface editor must integrate into the existing version of CAPS.
    1.2 The interface editor must run on a variety of platforms and operating systems besides SunOS UNIX.
    1.3 The GUI tools' licensing costs should be kept to a minimum.
    1.4 The execution of a CAPS prototype should not be adversely effected by the GUI repaint and input events.

1.5 The interface editor must be easy to use and be intuitive to a new user.
1.6 The interface editor should contain a variety of high-level widgets to aid in the development of user interfaces.
1.7 The interface editor should contain a GUI builder tool.
1.6.1 Using the GUI builder, the interface widgets can be added, deleted, and replaced.
1.6.2 The GUI builder should be able to create high-level widgets.
1.7 The interface editor should require little training.
1.8 The interface editor should be easy to use.
1.9 The GUI-prototype may or may not be part of the static schedule.

2. Constraints
2.1 The GUI integration will be incorporated into CAPS Version 1.1.
2.2 The GUI integration will be developed on a Sun Microsystems Sparc-10 running SunOS 4.1.4.

B. DESIGN RESULTS - GUI-CLIENT / PSDL-SERVER

When using CAPS, the GUI-prototype is for analysis of design, displaying data, and user interaction with the PSDL application. Alternatively, a CAPS designer may want the GUI-prototype to emulate an external environment. The GUI-prototype may not have real-time constraints or be required in a final embedded system, but it will require CPU resources. In this case, it is important that the GUI-prototype not interfere with the real-time execution of prototype application.

As a result, the new design de-couples the GUI-prototype from the rest of the prototype. The new design contains components from a multi-tier client/server architecture. The new design creates a *GUI-client* that communicates with a *PSDL-server*. A GUI-client is defined as a GUI-prototype, which is de-coupled from the PSDL and executes on a client processor. A PSDL-server is a PSDL application, without a GUI, which executes on a server processor. With this design, a user can create the GUI-client with any programming language and GUI development tools. The client/server architecture allows the GUI-client to be located on a local CPU, while the PSDL-server is running on a remote CPU (Figure 8). The GUI-client event model can be retained by running the PSDL-server on a different CPU. Also, the real-time schedule of the PSDL-

server is not affected by the GUI-client. In addition, the design allows the GUI-client to be compatible with existing CAPS Version 1.1 software.

The client/server communication is possible with a variety of middleware solutions. The simplest middleware solution is sockets.

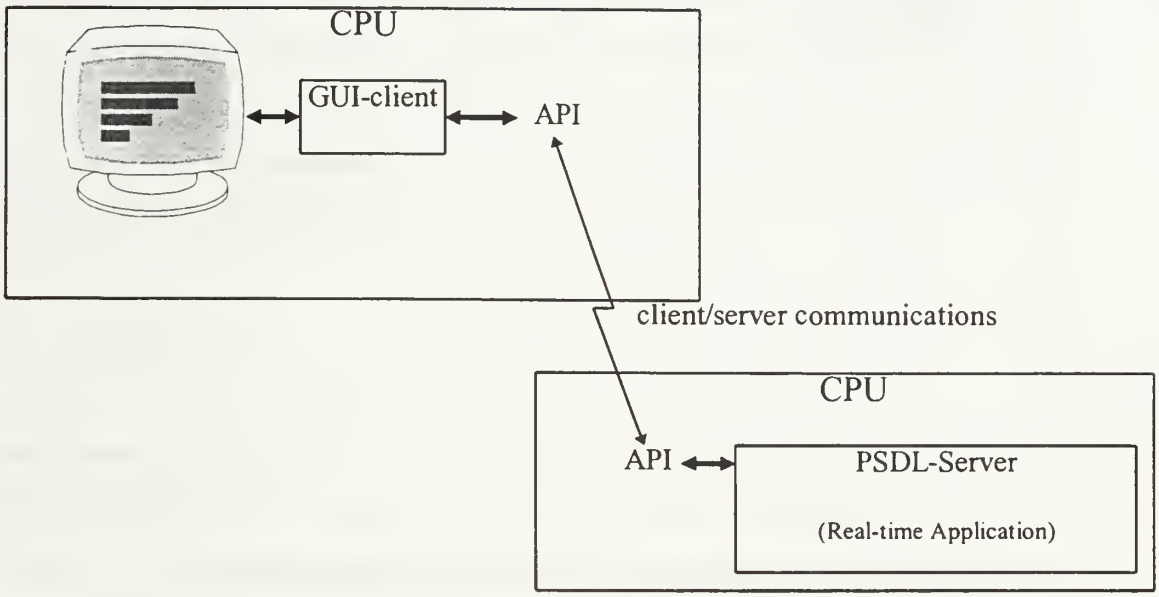

Figure 8. Multiple CPUs CAPS Client/Server

Alternatively, the client/server design could be implemented on one CPU if the processing of the GUI-client is taken into account. For this purpose, a prototype application could be run on one CPU by using synchronous communications between the GUI-client and the PSDL-server as shown in Figure 9. As a result, the synchronous communications could prevent the GUI-client from taking CPU resources during the execution of the PSDL-server.

Threads could be used to implement the one-CPU design. The first thread could execute the PSDL-server, while the second thread handles the GUI-client during non-critical times. Also, a single workstation with multiple CPUs could be evaluated.

Problems with the client/server design include: increased overhead and network traffic. A dedicated network will minimize the amount of network traffic. Another

problem with a CAPS client/server design is the added complexity in the overall CAPS system.

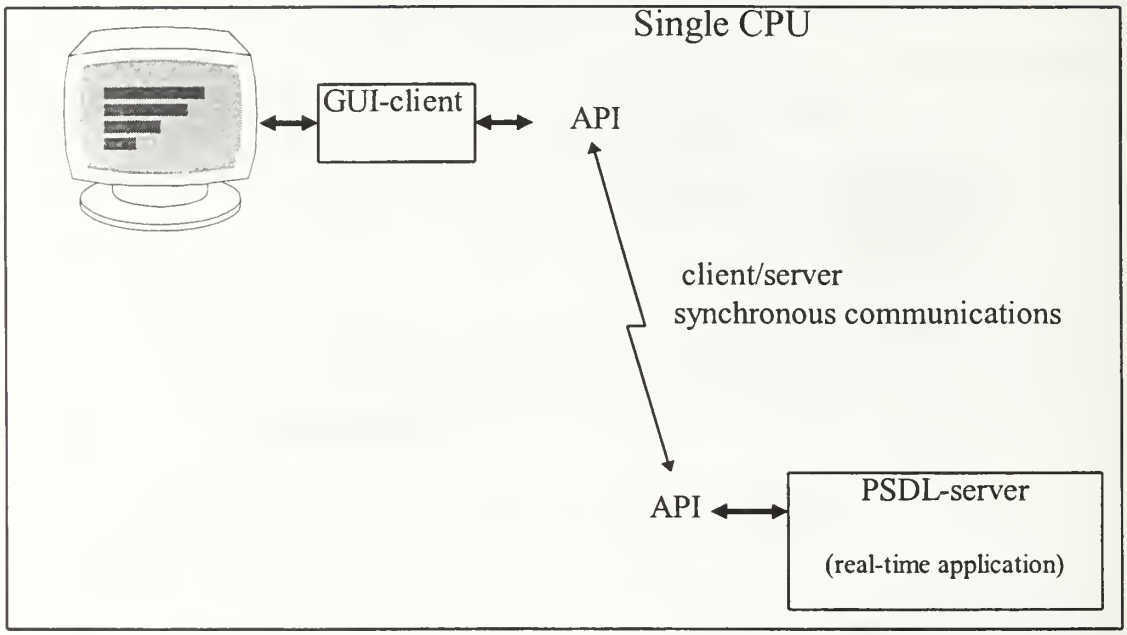

Figure 9. One CPU CAPS Client/Server

IV. IMPLEMENTATION CONSIDERATIONS

A. SELECTION OF GUI TOOLS

I spent approximately six months evaluating and selecting GUI development tools. There are over a hundred commercial tools available. In addition, many universities have departments that are doing GUI research, and they offer their GUI development tools free-of-charge. Appendix A describes the evaluated tools. The following tools are considered as possible choices for integration in CAPS.

1. Fresco

Fresco is an object-oriented development toolkit for the development of user interfaces[Ref. 27]. Fresco is a design evolution from Stanford University's Interviews toolkit. Fresco was developed at Fujitsu's Faslab in conjunction with the X-Consortium. The Opengroup's X-Windows11-V6 (Broadway) contains a sample implementation of Fresco. [Ref. 28]

Unfortunately, Faslab discontinued support and development work on Fresco in late 1996. Also, the software community has failed to widely use Fresco. For these reasons Fresco was not integrated into CAPS.

2. Java Development Kit

The Java programming language, developed by Sun Microsystems, is being hyped as a replacement for C and C++. The Java programming language is becoming the programming language of choice for many internet and standalone applications [Ref. 29][Ref. 30]. Java is not a GUI development tool, but only a language and toolkit. Fortunately, many vendors are now selling complete GUI development environments for Java. Additionally, many traditional Motif development tools are now supporting Java.

Java is an object-oriented, distributed, interpreted, secure, platform-independent, and multithreaded programming language. Unlike most languages, Java has automatic garbage collection. Java has client/server protocols included in the language's API. In addition, Java has a built-in Abstract Window Toolkit (AWT). The AWT can be considered a PIGUI because it runs on mulitple platforms without program modification. The Java AWT also supports the native look-and-feel of the target platform.

Consequently, many universities are now offering Java as the introductory programming language instead of C++. Appendix A contains more information about the Java programming language and development tools.

A GUI-client written in Java would allow the interface to run on almost any platform and communicate with a PSDL-server. Java was selected to test integration with CAPS.

3. Visual Workshop / X-Designer

Sun Microsystems' Visual Workshop and Imperial Software Technology's (IST) X-Designer are X-Windows/Motif tools for UNIX platforms. Visual Workshop incorporates IST's GUI builder (X-Designer) into its product. Both tools provide about the same functionality. The tools create Motif X-Windows code written in C and C++. Ada95 can be generated with OC-Systems' XDA attachment. Visual Workshop was also selected to test integration with CAPS. [Ref. 21][Ref. 31]

B. SELECTION OF CLIENT/SERVER MIDDLEWARE

The number of techniques for implementing client/server middleware is very large. Many vendors are trying to sell their products and create standards. The products include: Sockets, CORBA, CGI, Netscape's Caffeine, Java-RMI, and Microsoft's DCOM.

1. Sockets

A socket is the basic protocol for client/server communication over TCP/IP stacks. Sockets were originally introduced in 1981 for UNIX BSD 4.2. Sockets are available on almost every operating system. Sockets are available in several forms: *datagram*, *stream*, and *raw*. The most common socket API is the Berkley UNIX C protocol. Sockets can also be used for communication on a single computer. Figure 10 shows a typical sequence for establishing a client/server connection. Sockets are the basis for most higher level client/server middleware. The socket programming model is quite primitive, but it is very fast and a well-known standard. Sockets were implemented and evaluated as a possible middleware between the PSDL-server and the GUI-client. [Ref. 32][Ref. 33]

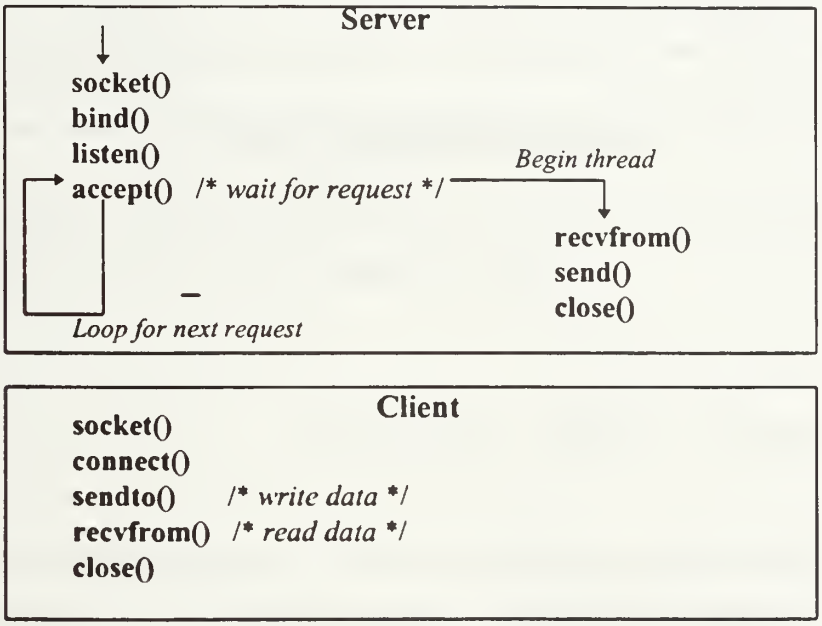

Figure 10. Client/Server Scenario

2. Common Object Request Broker Architecture (CORBA)

CORBA is a middleware project being developed by the Object Management Group (OMB) consortium, consisting of over 700 companies. The notable exception to

this group of companies is Microsoft, which has a competing product called the Distributed Component Object Model (DCOM). CORBA could replace all other implementations of client/server middleware. In CORBA client/server computing, objects cooperate over the network as opposed to cooperating processes. [Ref. 32] Specifications for CORBA objects are written with the Interface Definition Language (IDL). IDL provides operating system and programming language interfaces to other services on the CORBA bus. CORBA allows almost any programming language to communicate over a network as show in Figure 11.

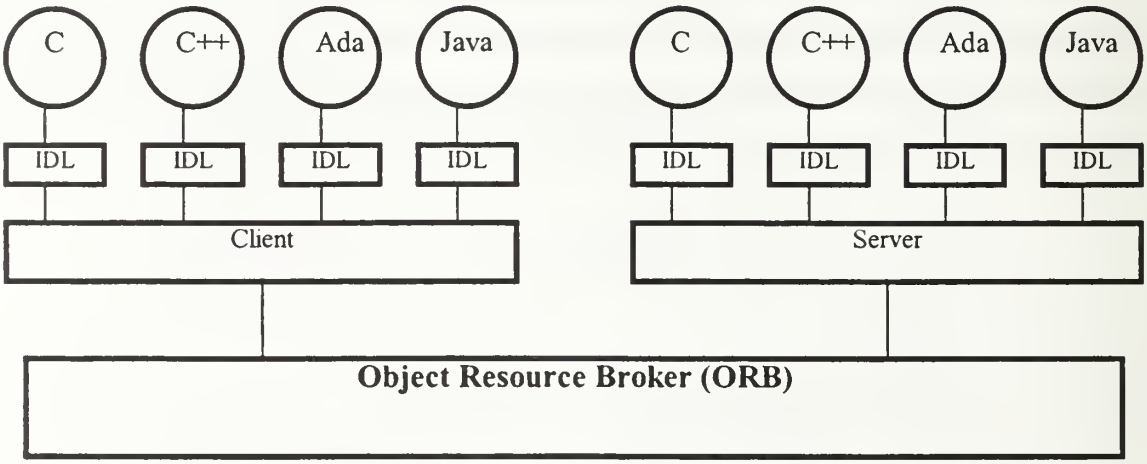

Figure 11. CORBA

C. PSDL-SERVER – SOCKET – GUI-CLIENT

In this implementation, A GUI-client, written in Java or Motif, communicates with a PSDL-server by using TCP/IP sockets. Figure 12 shows the implementation of the design. Demonstrations of the implementation are described in the next chapter. The source code for the demonstrations is in Appendix B.

The PSDL Ada atomic operators bind to a C module that implements the server side of the socket connection. The server socket is written is C because of the difficulty in implementing sockets with the VADS Ada83 compiler.

A protocol is established for passing a data structure between the GUI-client and the PSDL-server. An example communication data structure is shown below:

```
struct CapsData {
        char command_number;
        int i1
        int i2;
        char buffer[SIZE];
}
```

At present, the communications data structure is modified for each new application. A more generic data structure should be designed to work with all prototype applications. The client/server communications in the demonstrations are synchronous. Asynchronous communication may also be used, but care must be taken to avoid data-loss or deadlocks. The implementations are multithreaded. One thread in the GUI-client is devoted to the socket communication.

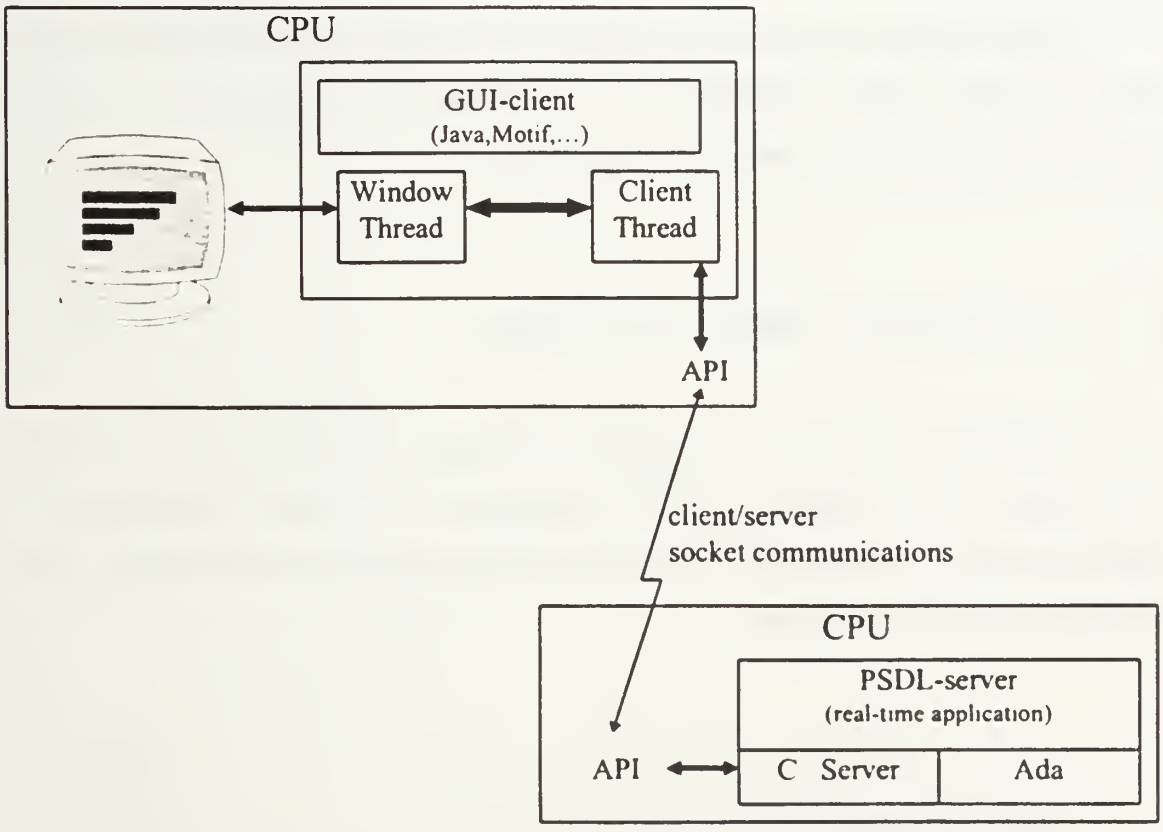

Figure 12. CAPS-Socket-GUI

1. Java GUI-client

The GUI-client was implemented using the Java programming language. The Java GUI-client can be created with the Java development kit or designed with development tools, such as Symantec's Visual Café [Ref. 34]. The Java GUI-client can either be a standalone application or run as an applet in a network browser. The Java implementation is multithreaded. Unfortunately, programming Java threads can be difficult. The *Thread Scheduling Model*, which is platform dependent, determines the thread that will be running at any given time. Some operating systems use *preemptive scheduling* and others use *time slicing*. Consequently, the programmer must carefully design the application so that it is still platform independent. The code for a Java implementation is in Appendix B.

2. Motif GUI-client

Another GUI-client was implemented in Motif using Visual Workshop (Version 3.) for a Sun Microsystems' workstation. The GUI-client is also multithreaded. The code for a Motif implementation is included in Appendix B.

D. PSDL-SERVER – CORBA – GUI-CLIENT

CORBA has the potential of making middleware transparent to the application programmer. In this thesis, the CORBA middleware was not fully implemented. A CORBA interface is written in IDL. An example IDL interface between the PSDL-server and the GUI-client is as follows:

```
// IDL
interface Capsdata {
        attribute int ix;
        attribute int iy
```

```
        int repaint()
        int readdata()
        int writedata();
}
```

V. DEMONSTRATIONS OF NEW DESIGN

A. ROBOT CONTROL SYSTEM

This demonstration reworks a CAPS project originally developed in CS4920 (Spring 1996). The project developed a software prototype for a robot control system. The prototype was developed using CAPS Version 1.1. This demonstration uses a socket to connect the GUI-client with the PSDL-server.

1. Requirements for Robot Control System

The robot moves on a friction-less table using air bearings, and is equipped four compressed air thrusters that are aligned with the directions of the coordinate axes (+x, -x, +y, -y). Opposing thrusters should never be both turned on at the same time. Thruster force can be continuously varied under computer control. The maximum thrust from each thruster produces an acceleration of 1 meter per second per second. The software is supposed to provide a "soft landing" capability for the robot. In the test bench for the robot control software, the initial positions (x and y coordinates of the robot) and initial velocities (x and y components of its speed) are specified as input. The controller is supposed to bring the robot to a stop at a position less than 2 cm from the origin of coordinate system. The robot must never get closer than 1 cm from the origin, and it must stay within the border of the table (-1 meter <= x <= 1meter and -1 meter <= y <= 1 meter). The initial position must be a legal position, and both components of the initial velocity must not exceed 1 meter per second.

2. Robot: PSDL-server

CAPS (Version 1.1), running on a Sun Microsystems' Sparc10 (SUNOS 4.14), was used to implement the demonstration. The PSDL for this demonstration is simple, but

contains time-critical operations. Appendix B contains the source code for the PSDL-server. Figure 13 shows the Robot PSDL graph. The operators *bop_display* and *bop_input* are atomic operators implemented in Ada. These operators bind to a C program that implements the server connection. The atomic operator *bop_display* sends data to GUI-client while *bop_input* requests input from the GUI-client.

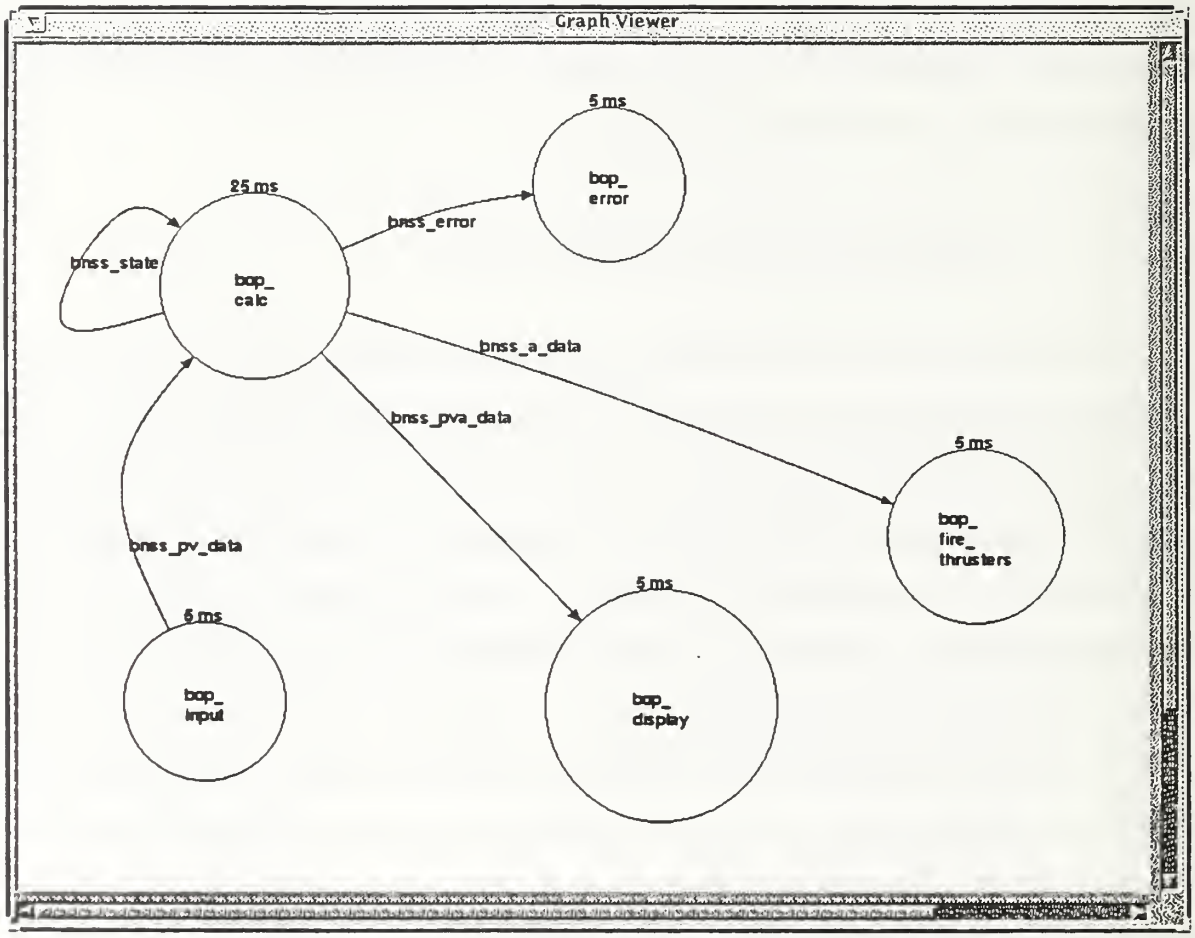

Figure 13. Robot PSDL Graph

Code fragments for the PSDL-server are shown below. The file ada.h contains the data structure that is passed between the GUI-client and PSDL-server. The file "bop_display.a" is source code for the atomic operator bop_display. This atomic operator passes display data to GUI-client. Inside "bop_display.a", the function call *socket( )* passes the data structure to the GUI-client. The file "socket.a" contains the source code

for *socket( ),* which binds to the C program "server.c" which performs the actual socket connection, socket binding, and data transfer.

```
-------------------------------------------------------------------
-- FILE       : ada.h
-------------------------------------------------------------------
typedef struct {
      int              ix;
      int            iy;
      int            ir;
} Rec, *Rec_Ptr;

-------------------------------------------------------------------
-- FILE       : socket.a
-------------------------------------------------------------------
--pkg
package body socket_PKG is
          a1 : FLOAT;
          b1 : FLOAT;
          type Rec is record
                    ix: Integer ;
                    iy: Integer ;
                    ir: Integer ;
          end record;
          type Rec_Ptr is access Rec;
          I : Rec_Ptr := new Rec;
          pragma LINK_WITH ("server.o");
          procedure server(I : in Rec_Ptr);
          pragma Interface (C, server );

      procedure Socket(xx: in out Float; yy: in out Float;
                    ireq: in out INTEGER) is
       begin
          .
          .
                    server(I);
          .
          .
       end Socket;
end socket_PKG;

-------------------------------------------------------------------
--FILE       : bop_display.a
-------------------------------------------------------------------
with pva_data_Pkg;
with socket_Pkg;
use socket_Pkg;

procedure bop_display(bnss_pva_data: in pva_data_Pkg.pva_data) is
    begin
.
```

```
.
    socket(fx,fy,ix);        -- Send the x,y position to the GUI-client.
.
.
 end bop_display;
```

3. Robot: GUI-Client

The GUI-client can be written with almost any language and GUI development tools. A GUI-client communicates with the PSDL-server.

a. Java

The Java GUI-client was developed on a Sun Microsystems' UltraSparc (Solaris 2.5) and a Windows95 PC, using the Java Developers Kit (Version 1.1). In addition, Symantec's Visual Café (Version 1.0) was used to develop the GUI-client. The Java program can be designed for a browser or as a standalone application. Figure 14 shows a Java-applet version of the GUI-client. The source code for the applet version is in Appendix B. The Java applet is being displayed in a Netscape browser. The Java program is multithreaded. One thread responds to read/write data via the socket, while the other thread waits for user input and paints the screen.

The Java program has various controls. The "Restart" button reinitializes the socket connection. The "Zoom" button changes the aspect of the view. The user can enter the initial x, y, x-velocity, y-velocity. The user can also drag the robot to a new starting location.

b. Visual Workshop

The GUI-client was also implemented using Sun Microsystems' Visual Workshop version 3.0. running on a Sun Microsystems' Sparcstation-20 (Solaris 2.5). The GUI-client is very simple, but it does contain a thread to read/write data from the

PSDL-server. Figure 15 shows the Visual Workshop GUI builder, with the GUI-client loaded. Figure 16 shows the Visual Workshop version of the GUI-client.

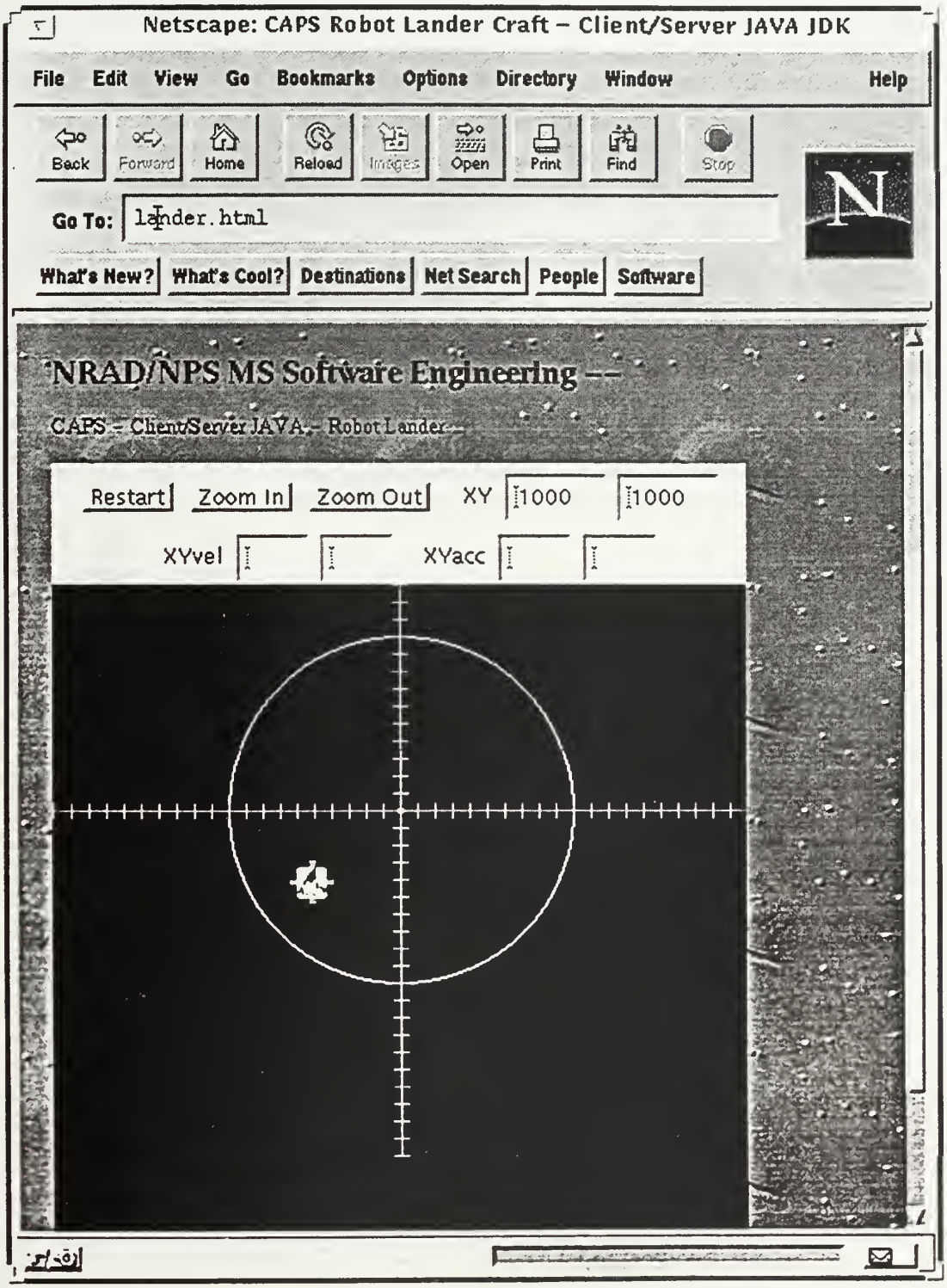

Figure 14. Java GUI-Client

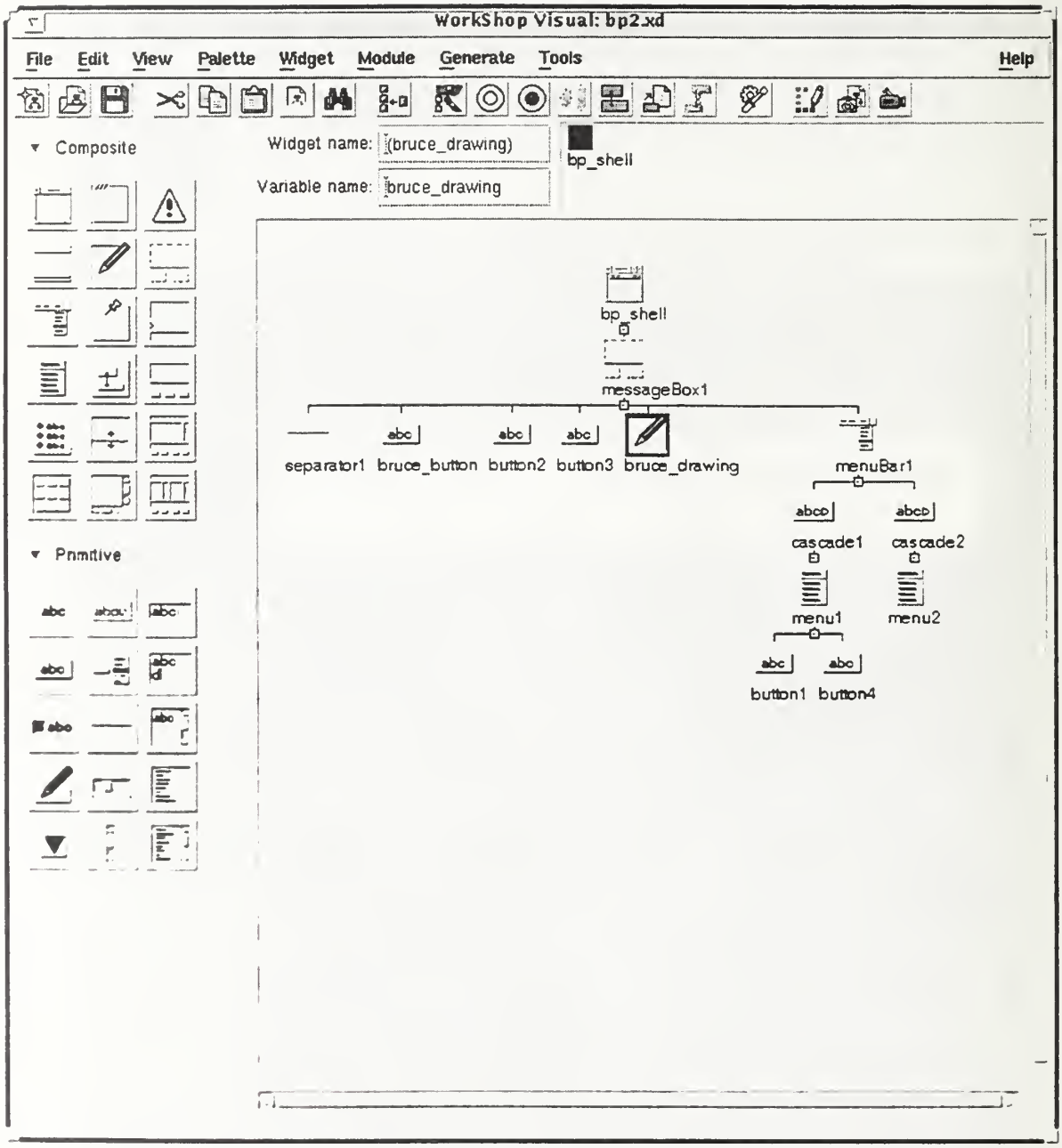

Figure 15. Visual Workshop GUI builder

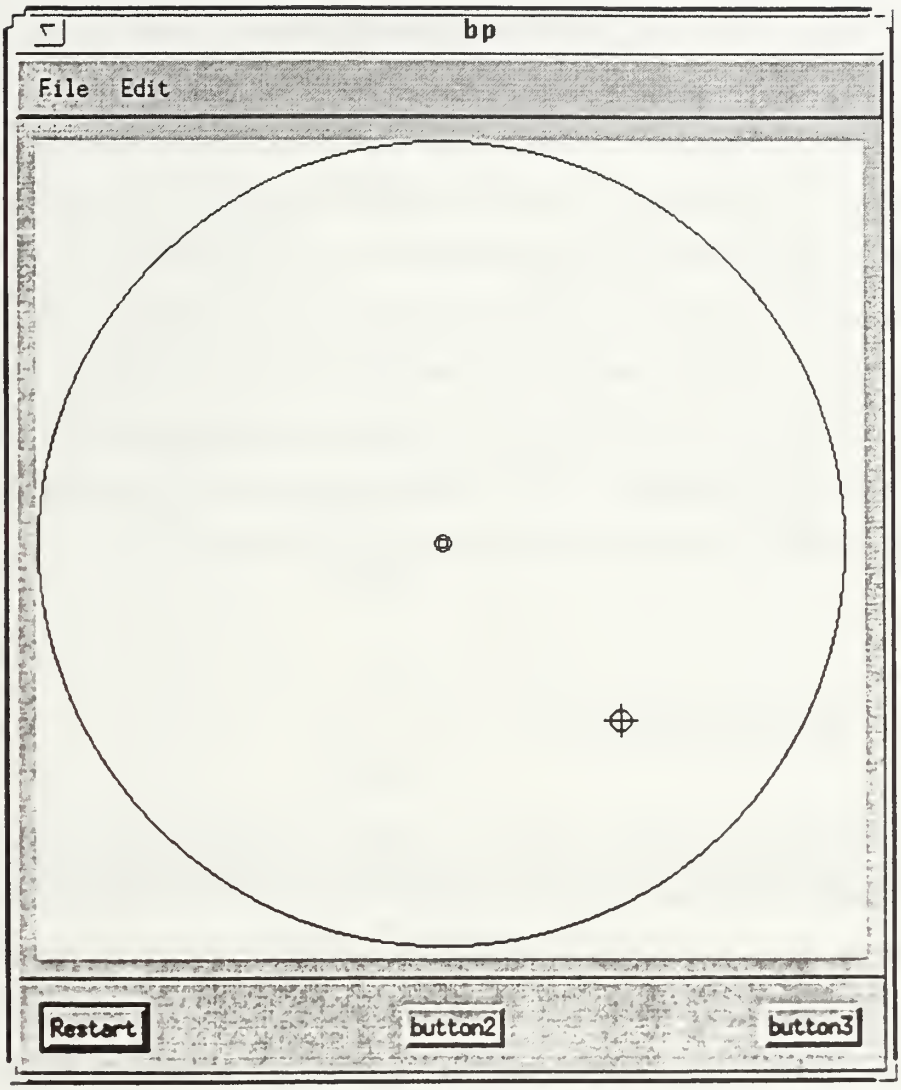

Figure 16. Visual Workshop GUI-Client

B. SMARTSHIP PROJECT

The Smartship Project commenced in December 1995 in response to the CNO's review of the 1995 Summer NRAC study which focused on reduced manning. The study concluded that major reductions in manning could be achieved by design in new construction ships. Commander Naval Sea Systems Command responded to the CNO's desire to focus on existing operational ships. The overall goal is to reduce workload, improve mission readiness, and maintain safety at minimum cost to the government. A Naval Postgraduate School thesis "Smartship Project Modeling", by Nolan Ruiz, is trying to determine if an effective modeling approach for the Smartship project can be found. In addition, it tries to determine if the current *Productivity Allowance* is valid. [Ref. 35]

This thesis will look at possible user interfaces, using the GUI-client/PSDL-server design, that could be used with the Smartship project. A socket is usedto connect the GUI-client with the PSDL-server. This demonstration will contain only preliminary results, but Nolan Ruiz's thesis will contain a complete prototype.

1. Smartship PSDL-server

The Manpower/Workload profile of a ship's Communications Division was evaluated. Tasks can either be periodic or sporadic. Periodic tasks include: Station Watch, PMS Actions, and Field Day. Sporadic tasks include: Underway Replenishment, Vertical Replenishment, General Quarters Drills, Fire Drills, Security Drills and Training.

CAPS (Version 1.1), running on a Sun Microsystems' Sparc10 (SUNOS 4.14), was used to implement the CAPS demonstration. The top-level PSDL graph contains the following composite operators: SHIP, NAVSEA, and USER. The SHIP composite operator can be broken down into periodic and sporadic tasks. Figure 17 is the PSDL graph for the SHIP composite operator. The data flow between the operators (the periodic and sporadic tasks) is in man-hours.

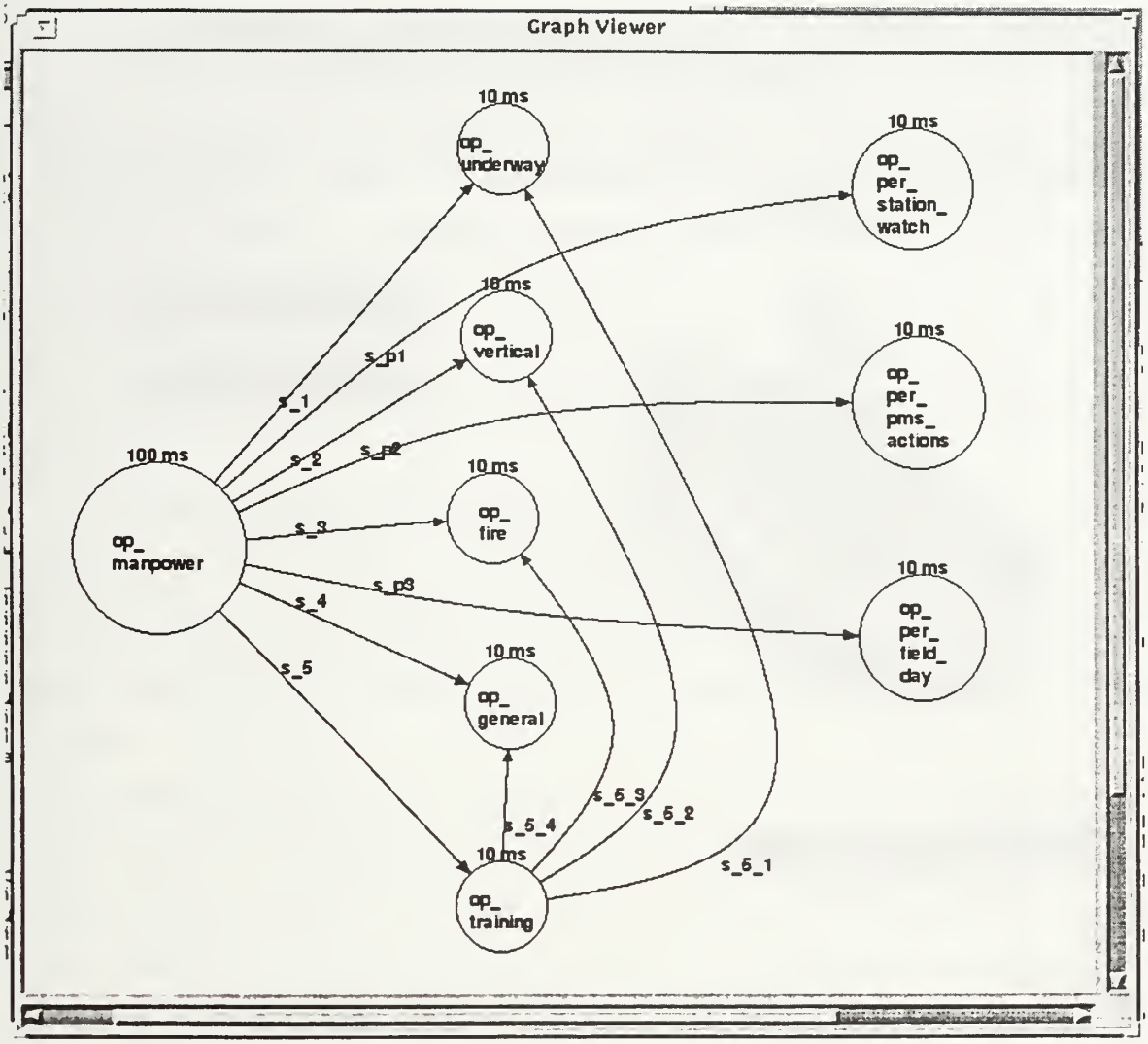

Figure 17. Smartship PSDL

2. Smartship GUI-client

The Java GUI-client was developed on a Sun Microsystems' UltraSparc (Solaris 2.5) and a Windows95 PC, using the Java Developers Kit (Version 1.1). In addition, Symantec's Visual Café (Version 1.0) was used to develop was used to develop the GUI-client. Figure 18 is a Java version of a GUI-client for the Smartship prototype.

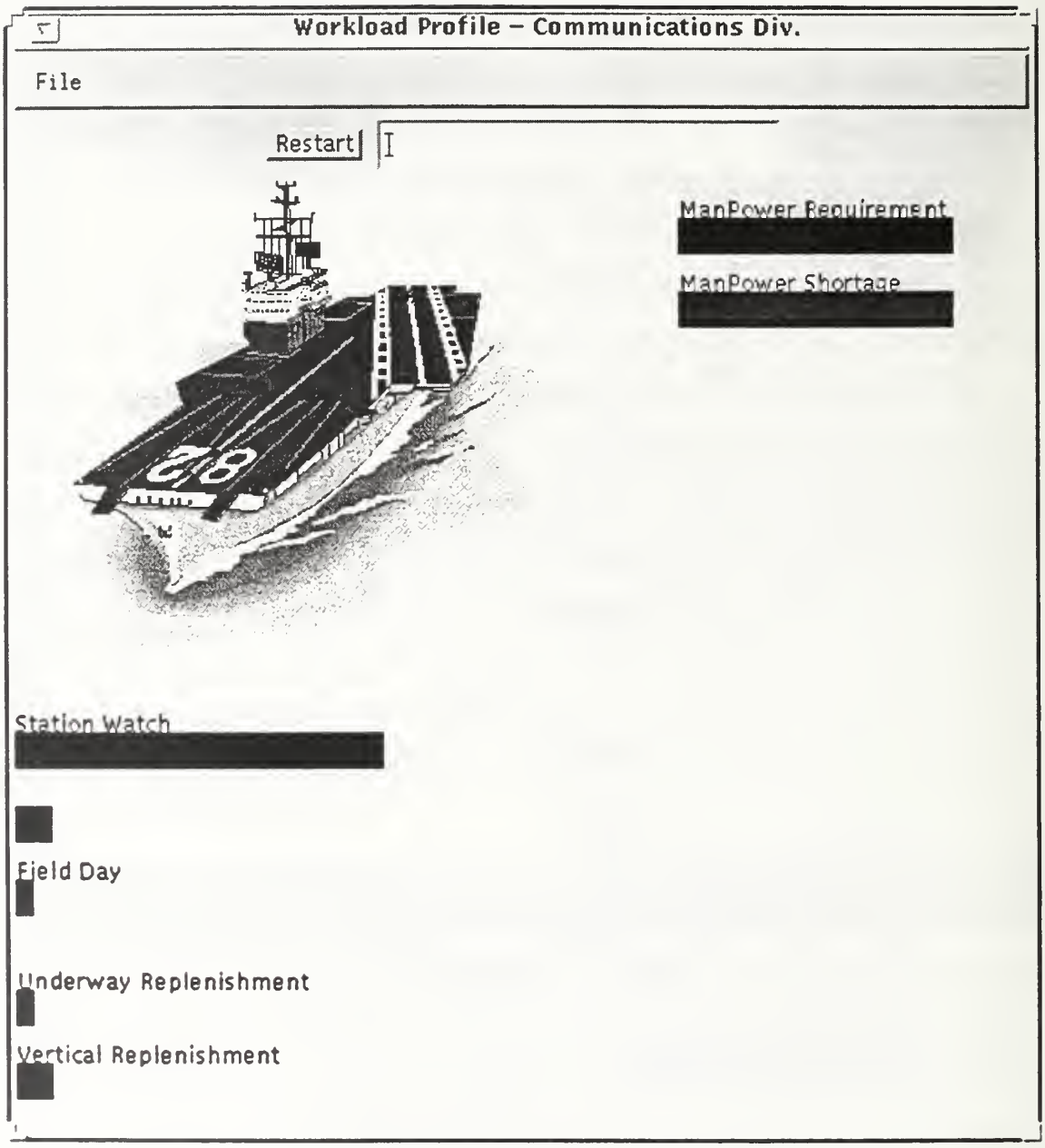

Figure 18. Smartship GUI-client (Java)

VI. CONCLUSION

A. SUMMARY OF DESIGN AND IMPLEMENTATION

The client/server design allows a PSDL-server to communicate with a GUI-client. The design is successful in decreasing the coupling between the GUI-client and the real-time prototype application. The GUI-client can be created with almost any GUI development tools. In addition, the design allows the GUI-client to be located on a local or remote CPU. The GUI-client can be unbound or bound to the real-time schedule of the PSDL server.

The demonstrations, using a variety of commercial products, show the successful implementation of the design. In addition, the proper use of threads and GUI-client/PSDL-server communications (synchronous/asynchronous) is very important. If implemented properly, the client/server middleware does not effect the performance of the prototype.

Although this thesis did not finish the integration of a new interface editor in CAPS, many GUI development tools were evaluated for possible integration. The client/server design increases the number of GUI development tools that can be used in the CAPS environment. The Java development tools were determined to be the best tools available to use with CAPS. The reasons to use Java include: low cost, abundance of tools, popularity, language features, and platform independence.

The largest obstacle for implementation of the design is the client/server middleware. For a first implementation, sockets were able to provide the required functionality, but the socket programming model is very primitive. Therefore, a higher level product must be chosen. CORBA, with Ada, C, C++, and Java interfaces, could be used to implement the client/server middleware.

B. FUTURE RESEARCH

In the course of work on this thesis, many ideas came forward, but were not implemented. A great deal of time was spent on the evaluation of GUI tools and middleware. The following is a list of ideas for future work.

1. Selection of Middleware

In addition to sockets, other middleware should be investigated. CORBA was only partially investigated in this thesis, but it could be used to connect atomic operators written in Ada to a GUI-client. Other middleware to investigate include: Java's RMI, Microsoft's DCOM, Netscape's Caffeine, RPC.

2. Selection of GUI Development Tools

A GUI development tool should be selected for full integration into CAPS. In addition, a PIGUI would aid in the porting of the other CAPS tools.

3. Further Demos

New applications using the new GUI-client/PSDL-server design should be demonstrated. Previous CAPS applications could be ported and their performance compared. Implementations using a single CPU could be evaluated.

4. Socket Communications

If sockets are used, the server-side and the client-side source code could be automatically generated. The data structure for passing data between the GUI-client and PSDL-server could be standardized.

Several protocols exist for socket communications. The socket protocols should be compared to find the most efficient method of communications.

This thesis implemented the PSDL-server by binding the Ada83 code to a C module. The C module implemented the socket communication. Ada95 should be able to implement a socket connection.

5. JavaBeans

A *bean* is a reusable and interchangeable Java component. Even though JavaBeans are a new technology, many vendors are now selling JavaBeans tools. It might be possible to create a Java-Bean GUI-client automatically in the CAPS Graphics Editor (GE).

APPENDIX A. EVALUATION OF GUI TOOLS

A. GUI TOOLKITS

Product Name:	Java Development Kit
Vendor:	Sun Microsystems 2550 Garcia Ave. Mountain View, CA 94043 (800) USA-4SUN http://www.javasoft.com
Type of Tool:	Language, Library
PIGGUI:	Yes
Supported Platforms:	Window95/Nt, Macintosh, UNIX
Cost:	Free
Vendor Support:	Yes
Future Support:	Lots of current development
Languages Supported:	Java
GUI builder available:	No - See third party developers
Comments:	Java is an object-orientated, multithreaded, and portable programming language. The Abstract Window Toolkit (AWT) is a built-in library for creating window components. Java has built-in garbage collection. A great deal of work is being on and with Java, but the language is in a state of flux. The class library is very comprehensive, but lacks the high level GUI object or dialog objects. Many third parties are developing these high level objects. Unfortunately, Java is currently slower in execution speed than other languages, but Sun Microsystems and other vendors are developing compilers that should bring the performance of Java close to other languages. The loading of the Java AWT software on a Sun Microsystems Sparcstation (Solaris 2.5) and Windows-95 was very easy. Linking directly with CAPS would be difficult. A client/server interface with CAPS was completed in this thesis. [Ref. 29] [Ref.30] [Ref. 36] [Ref. 37]

Table 1. Java Development Kit

Product Name:	Fresco
Vendor:	Fujitsu FASLAB 800 El Camino, Suite 150 Menlo Park, CA 94025 (415) 325-6015 http://www.faslab.com
Type of Tool:	Toolkit, Similar look-and-feel of Motif
PIGUI:	Yes
Supported Platforms:	Windows95/NT, UNIX, Linux
Cost:	Free
Vendor Support:	None, the vendor has completed the research
Future Support:	The research on Fresco at FASLAB has be completed. No future work seems to be planned.
Languages Supported:	C++
GUI builder available:	No
Comments:	The Opengroup's X11-V6 (Broadway) contains a sample implementation of Fresco. Fresco is user interface system specified using CORBA IDL. Even though it is included in the latest version of X-Windows, Fresco is not yet a standard. Fresco is a PIGUI, but it does not support the native look and feel. Loading Fresco on Sun Microsystems Sparcstation (Solaris 2.5) was fairly simple. A tutorial walks a new user through examples. FrescoVFX is a fairly complete example of what Fresco can do. Fresco has very little low level documentation and the software library still seems buggy. There is no documentation on how to put the components together into a final product. Fresco could possibly connect to CAPS via a client/server approach. [Ref. 27]

Table 2. Fresco

Product Name:	wxWindows
Vendor:	Artificial Intelligence Applications Institute University of Edinburgh 80 South Bridge Edinburgh Scotland Phone: 0131 650 2746 http://web.ukonline.co.uk/julian.smart/wxwindows.com
Type of Tool:	toolkit library
PIGUI	Yes
Supported Platforms:	Windows/UNIX(Solaris)
Cost:	Free
Vendor Support:	None
Future Support:	?
Languages Supported:	C++
GUI builder available:	Yes, wxBuilder, Sun Microsystem's Devguide
Comments:	WxWindow is a C++ user interface library. It appears to be one of the best toolkits that is free. Loading wxWindows on a Sun Microsystems Sparcstation (Solaris 2.5) was difficult. The GUI builders are not very well documented. The class libraries have a lot of features, but learning all the classes and methods is difficult. CAPS could interface to wxWindows via a client/server protocol, but linking wxWindows into CAPS would be very difficult [Ref. 13]

Table 3. WxWindows

B. GUI BUILDERS

Product Name:	UIMX
Vendor:	Black & White Software (Resailer) 2155 S. Bascom Ave., Campbell, CA, 95008, 408-369-7400 http:/www/vedge.com
Type of Tool:	GUI-Builder (Motif)
PIGUI:	No
Supported Platforms:	UNIX
Cost:	~$5000
Vendor Support:	yes
Future Support:	yes
Languages Supported:	C, C++, Ada (add-on product)
GUI builder available:	Yes
Comments:	UIMX (Figure 19) is a very popular GUI builder. It is resold under many different vendors. Integrated products can include: ORBIX for client/server, Cross-platform toolset (windows), Ada. [Ref. 38]

Table 4. UIMX

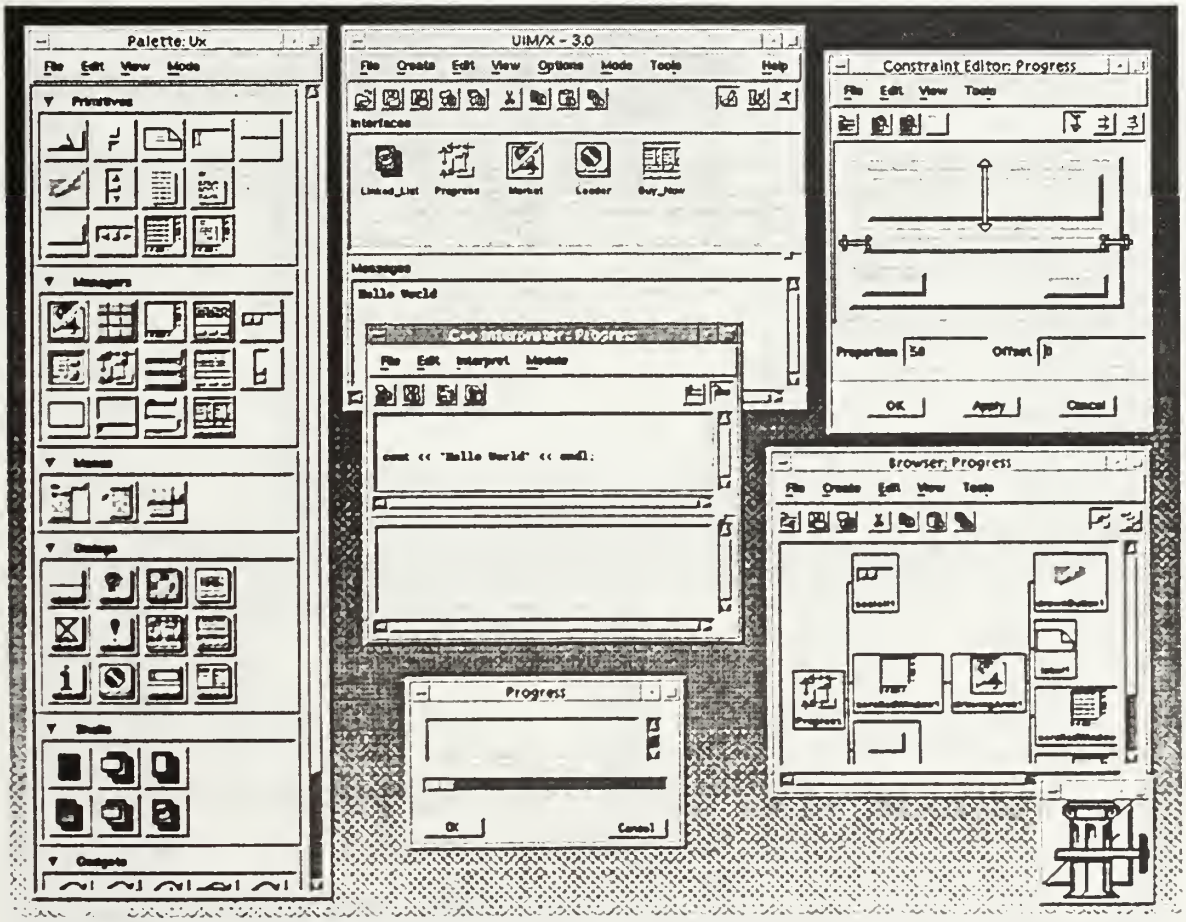

Figure 19 UIMX "From Ref. [38]"

Product Name:	X-Designer
Vendor:	Imperial Software Technology US Office 120 Hawthorne Avenue Suite 101 Palo Alto, California 94301 (415) 688-0200 http://www.ist.co.uk
Type of Tool:	GUI-builder (Motif, Java)
PIGUI:	No, Yes for Java
Supported Platforms:	UNIX platforms are supported
Cost:	~$3000
Vendor Support:	Yes
Future Support:	Yes
Languages Supported:	C, C++, Ada (with XADA), Java
GUI builder available:	Yes
Comments:	X-Designer (Figure 20) is a Motif development environment. With OC-Systems' XDA, it is possible to convert the output of Visual-Workshop to Ada. X-Designer is repackaged in many other 3rd party applications (Visual Workshop and DataViews). [Ref. 31]

Table 5. X-Designer

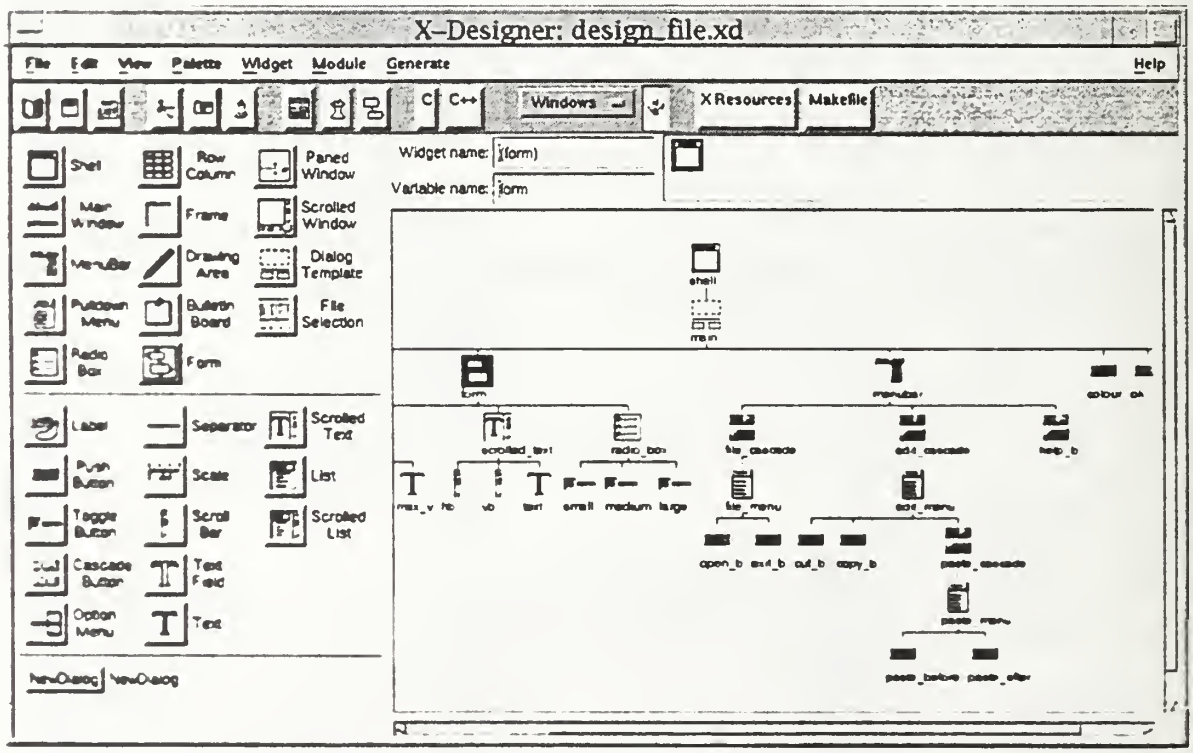

Figure 20. X-Designer "From Ref. [31]"

Product Name:	Visual Café
Vendor:	Symantec Corporation 10201 Torre Ave Cupertino, CA 95014 (408) 253-9600 http://symantec.com
Type of Tool:	GUI-builder, (Java AWT)
PIGUI:	Yes
Supported Platforms:	Winows95/NT, Macintosh
Cost:	$199
Vendor Support:	Yes
Future Support:	Yes
Languages Supported:	Java
GUI builder available:	Yes
Comments:	Visual Café (Figure 21) is development environment for Java.. Loading on Window-95 was very easy. Symantec has a variety of products, including tools to link Java to databases. Linking directly with CAPS would be difficult. A client/server interface with CAPS was completed in this thesis.[Ref. 34]

Table 6. Visual Café

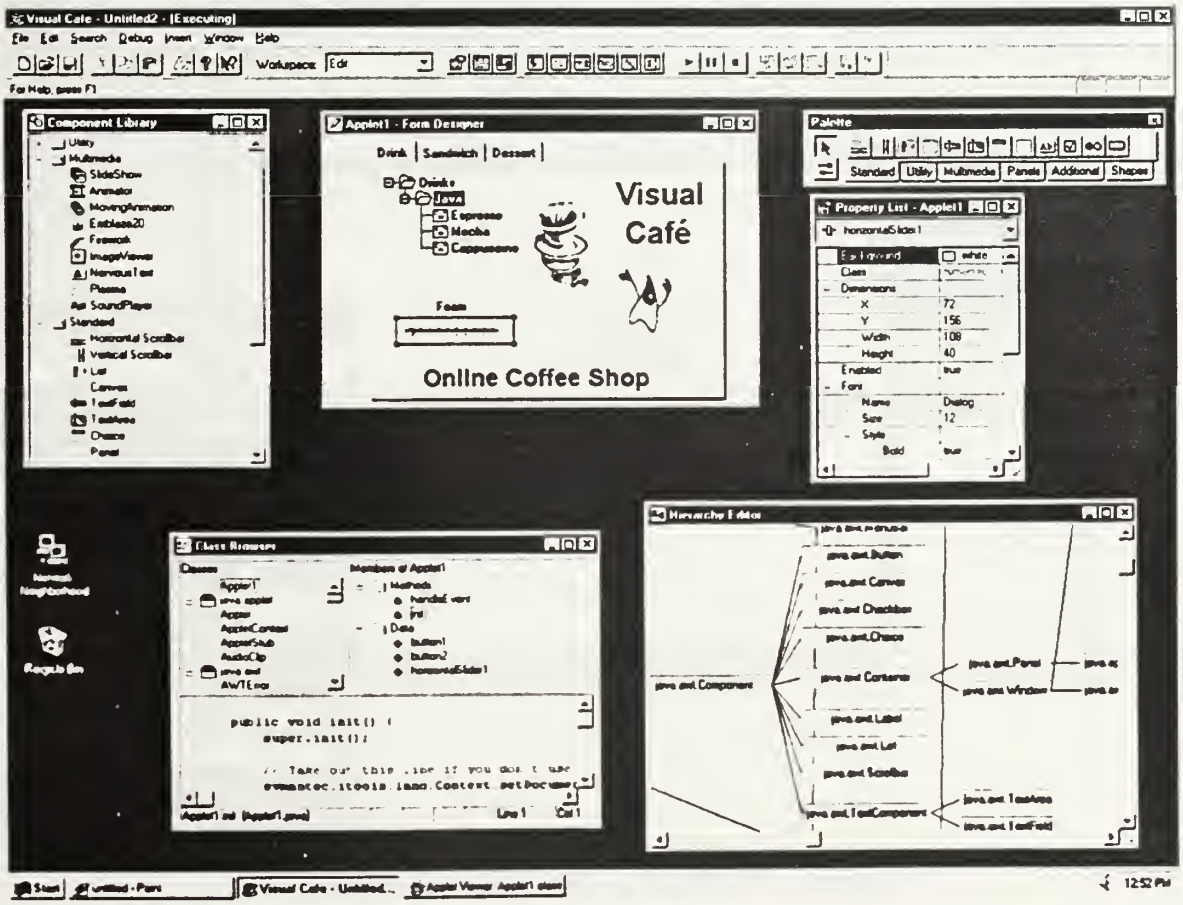

Figure 21. Visual Café "From Ref. [34]"

Product Name:	Builder Xcessory
Vendor:	Integrated Computer Solutions Cambridge, Mass. (617) 621-0060 http://www.ics.com
Type of Tool:	Motif and Java Gui Builder
PIGUI:	No (Motif), Yes (Java)
Supported Platforms:	UNIX: DEC,HP,IBM,SGI,SUN
Cost:	$3500
Vendor Support:	Yes
Future Support:	Seems Good
Languages Supported:	Java,C,C++,Ada
GUI builder available:	YES
Comments:	Builder Xcessory (Figure 22) is a popular GUI builder. Version 4.0 supports code generation for Motif and Java (applets and applications). The tool can be integrated into Pure-Atria's ClearCase evolution control products. There is support for Ada in Version 3. The support for Ada is limited to Sun Microsystems platforms and only with the Rational and SunAda compilers. [Ref. 39]

Table 7. Builder Xcessory

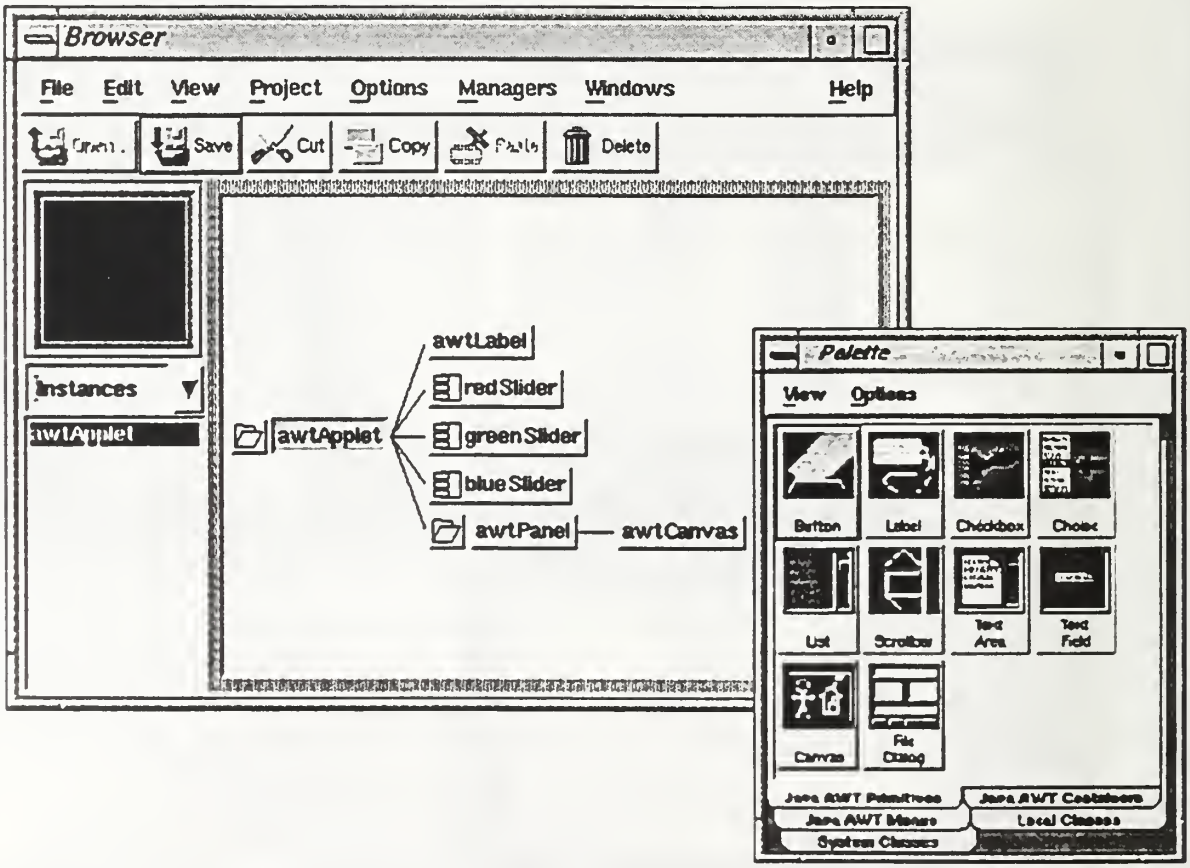

Figure 22. Builder Xcessory "From Ref. [42]"

C. USER INTERFACE MANAGEMENT SYSTEMS

Product Name:	OpenUI
Vendor:	Open Software Associates 20 Trafalgar Square Nashua, NH 03063 (800) 441-4330 http://www.osa.com
Type of Tool:	UIMS
PIGUI:	Yes
Supported Platforms:	Winows95/NT, UNIX, Linux
Cost:	~$4000
Vendor Support:	Yes
Future Support:	?
Languages Supported:	C, C++
GUI builder available:	Yes
Comments:	Open Software Associates main office is in Australia, but there support of their product is very good. OpenUI (Figure 23) has built in client(GUI)/server(LOGIC) application development. A user one must learn their interface language (OPL) to complete a user interface. C, C++ is used to bind to application code. ADA is longer supported. OpenUI would be useful for large project. [Ref. 15][Ref. 40]

Table 8. OpenUI

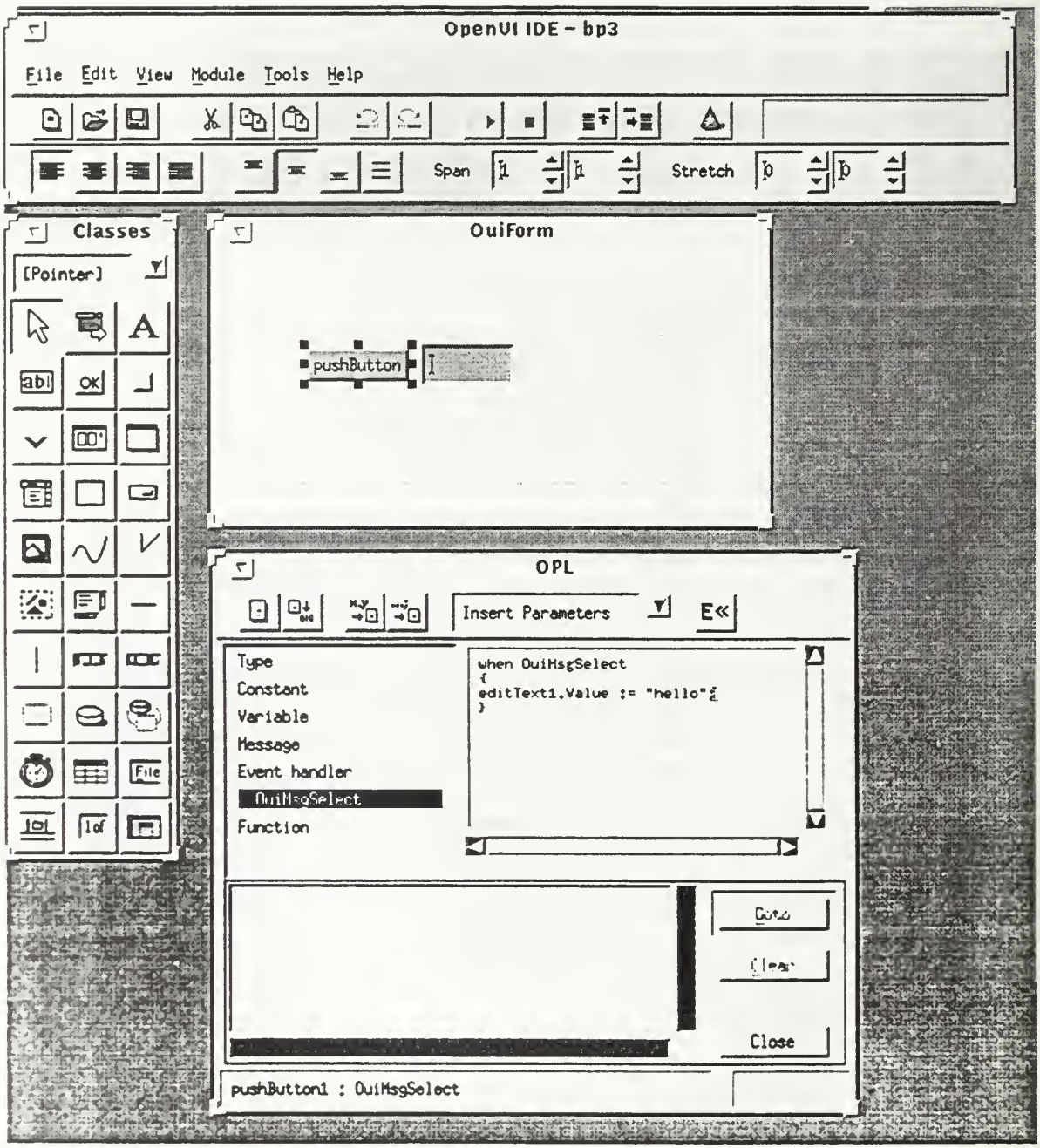

Figure 23. OpenUI "From Ref. [43]"

D. APPLICATION DEVELOPMENT ENVIRONMENTS

Product Name:	Visual-Workshop 3.0
Vendor:	Sun Microsystems 2550 Garcia Ave. Mountain View, CA 94043 (800) USA-4SUN http://www.sun.com
Type of Tool:	GUI-builder / Development Environment (Motif)
PIGUI:	NO
Supported Platforms:	Solaris2, Other UNIX platforms are supported with Imperial Software Technology's X-Designer.
Cost:	~$3000
Vendor Support:	Yes
Future Support:	Sun Microsystems is a leader in this market.
Languages Supported:	C, C++, Ada (with XADA)
GUI builder available:	Yes
Comments:	Visual-Workshop (Figure 24) is a Motif development environment that includes Imperial Software Technology's X-Designer (GUI-builder). Visual-Workshop is a complete engineering environment with a GUI builder, debugging tools, and evolution control tools. Installation of the software was very easy. A demonstration version can be downloaded and evaluated. The GUI builder is very easy to use. Linking directly with CAPS would be difficult, because of the event manager. A client/server interface with CAPS was completed in this thesis. With OC-Systems' XDA, it is possible to convert the output of Visual-Workshop to Ada. [Ref. 21][Ref. 41][Ref. 42][Ref. 43]

Table 9. Visual Workshop

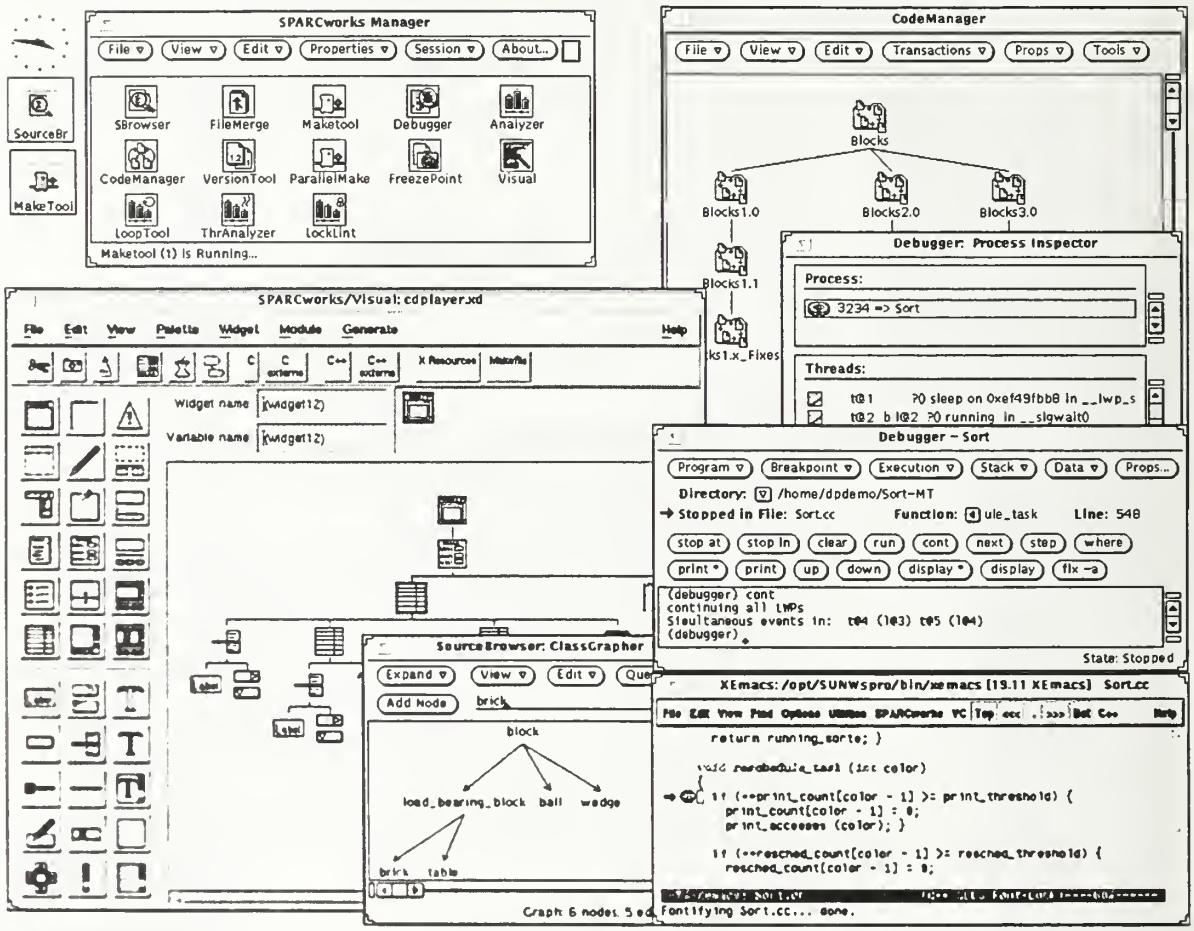

Figure 24. Visual Workshop "From Ref. [21]"

Product Name:	Galaxy
Vendor:	Visix 11440 Commerce Park Dr. Reston, Virgina 22091 (800) 832-8668 http://www.visix.com
Type of Tool:	Application Development Environment, GUI-builder
PIGUI:	Yes
Supported Platforms:	Win/NT, UNIX
Cost:	$9600
Vendor Support:	Yes
Future Support:	Yes
Languages Supported:	C, C++
GUI builder available:	Yes
Comments:	Galaxy (Figures 25, 26) is complete software engineering environment. Galaxy is an emulated API. An emulated API does not require high level toolkits to compile the program. For example, when Galaxy emulates Motif, Galaxy does not have to link in Motif libraries. A benefit of the emulated approach is that you can try out a Macintosh look-and-feel on a UNIX workstation. It also has a full range of other tools including: image and color editors. [Ref. 20]

Table 10. Galaxy

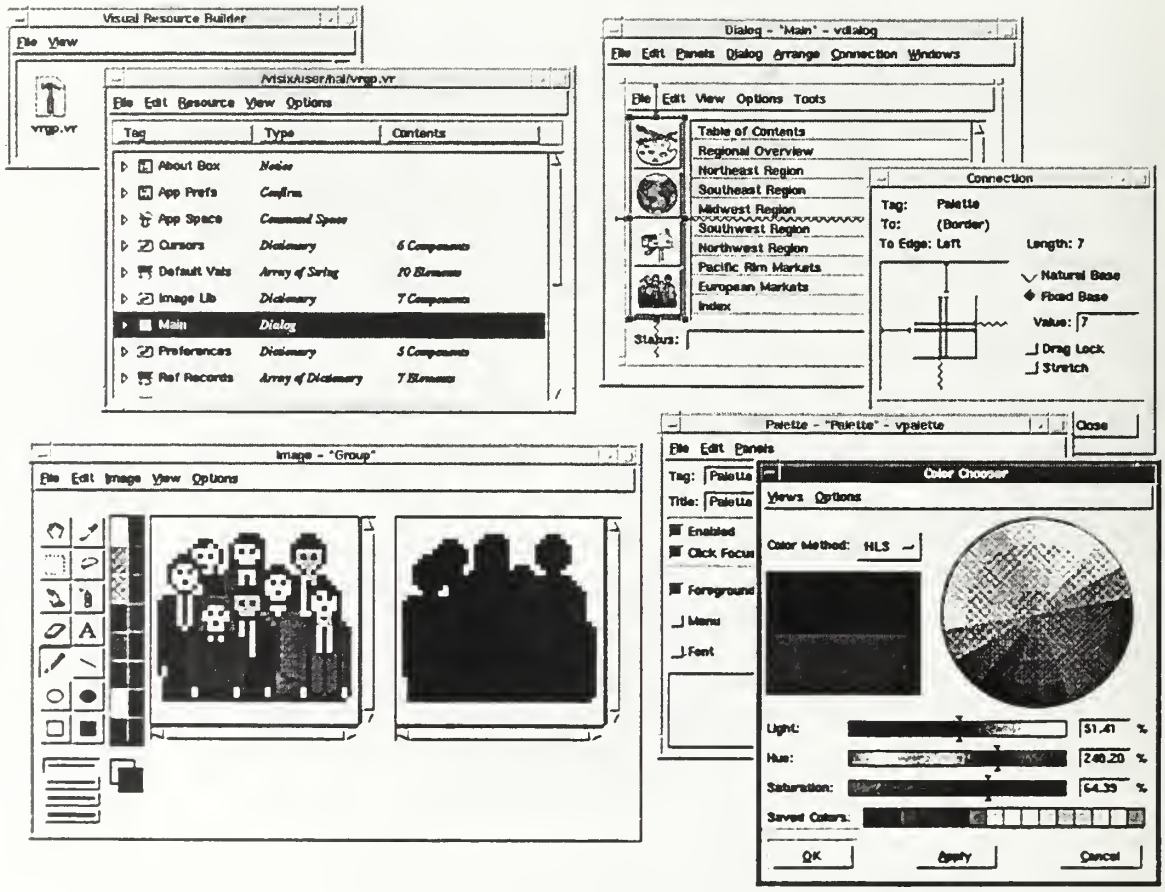

Figure 25. Galaxy (Motif) "From Ref. [20]"

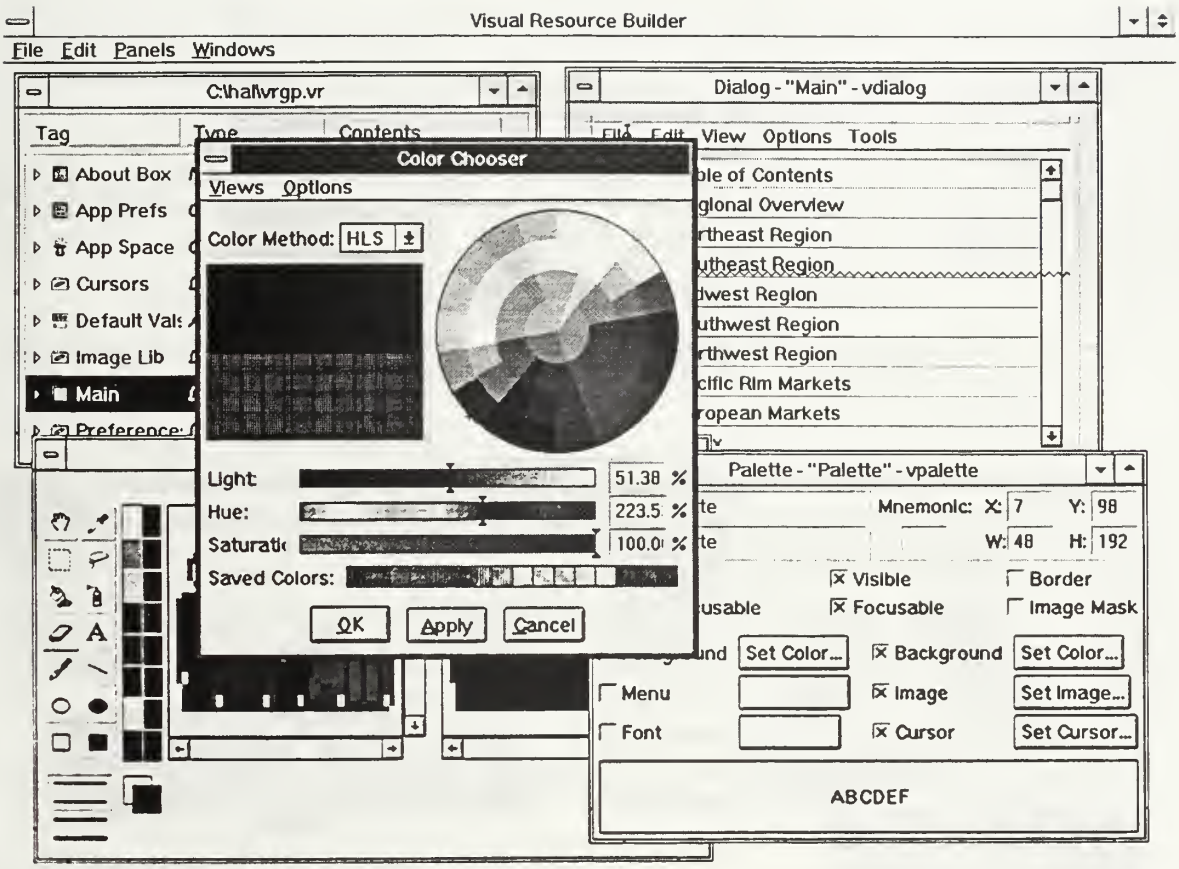

Figure 26. Galaxy (Windows) "From Ref. [20]"

E. OTHER GUI TOOLS

Product Name:	DataViews
Vendor:	DataViews Corporation (formerly VI Corp) 47 Pleasant St., Northampton, MA 01006 (413) 586-4144 http://www.dvcorp.com
Type of Tool:	Data Visualization for Real-time applications (Motif)
PIGUI:	?
Supported Platforms:	UNIX- Deployable on (VMS, Window-NT, Window3.1)
Cost:	~$18,000
Vendor Support:	yes
Future Support:	
Languages Supported:	C, C++, Fortran, Pascal
GUI builder available:	yes, an enhanced version of X-Designer
Comments:	Dataviews (Figure 27) is a data visualization tool for real-time applications. [Ref. 19]

Table 11. Datavews

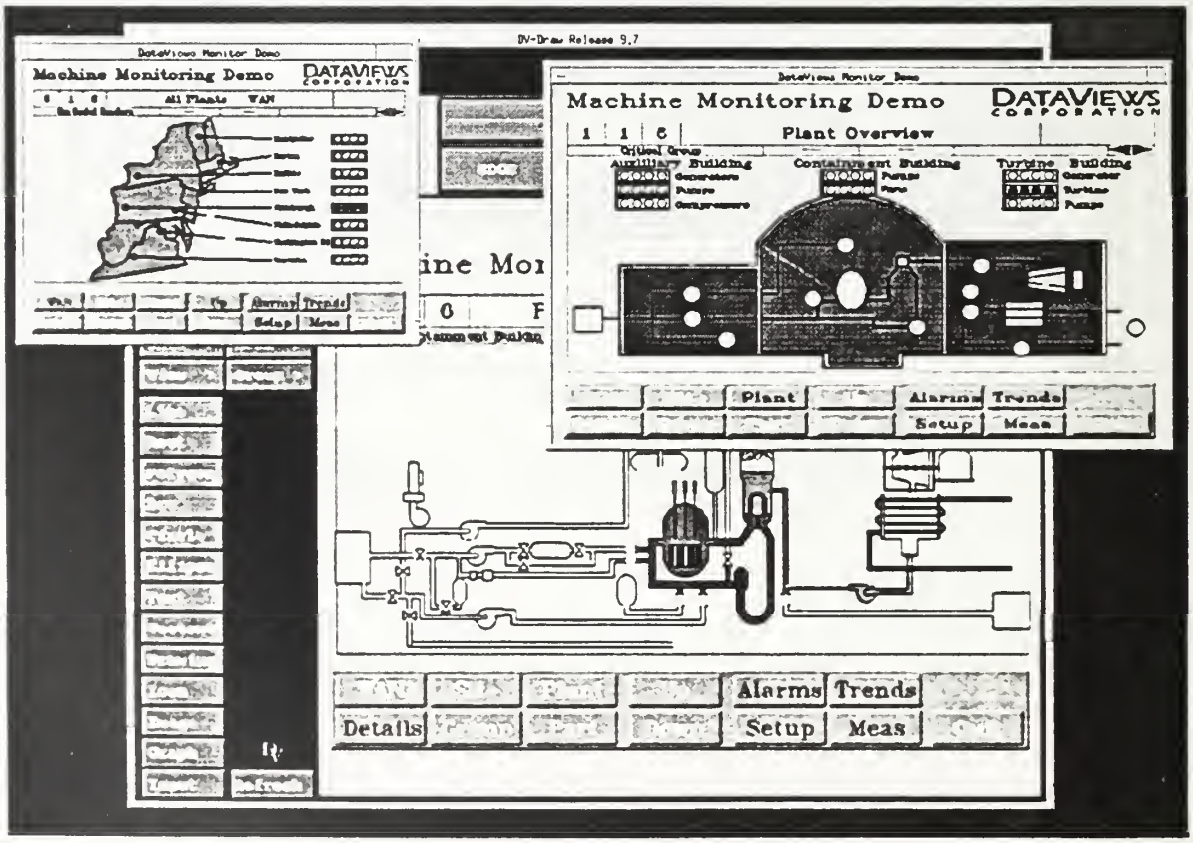

Figure 27. DataViews "From Ref. [19]"

Product Name:	Matlab
Vendor:	The MathWorks, Inc. 24 Prime Park Way, Natick, Mass, 01760 (508) 653-1415 http://www.mathwork.com
Type of Tool:	Numeric Computation, Graphing, and Visualization
PIGUI:	Partially, many programs can be run on different platforms with no change.
Supported Platforms:	UNIX, Window95/NT, Macintosh
Cost:	Varies on Platforms and options
Vendor Support:	Yes
Future Support:	Yes
Languages Supported:	Script Language, can be bound to C
GUI builder available:	Yes
Comments:	Matlab (Figure 28) is a popular signal processing and visualization tool. Additional toolboxes can be added Matlab. GUIs can be easily created in MATLAB. The Matlab scripting language is easy to use and there are many example of source code available. [Ref. 18]

Table 12. Matlab

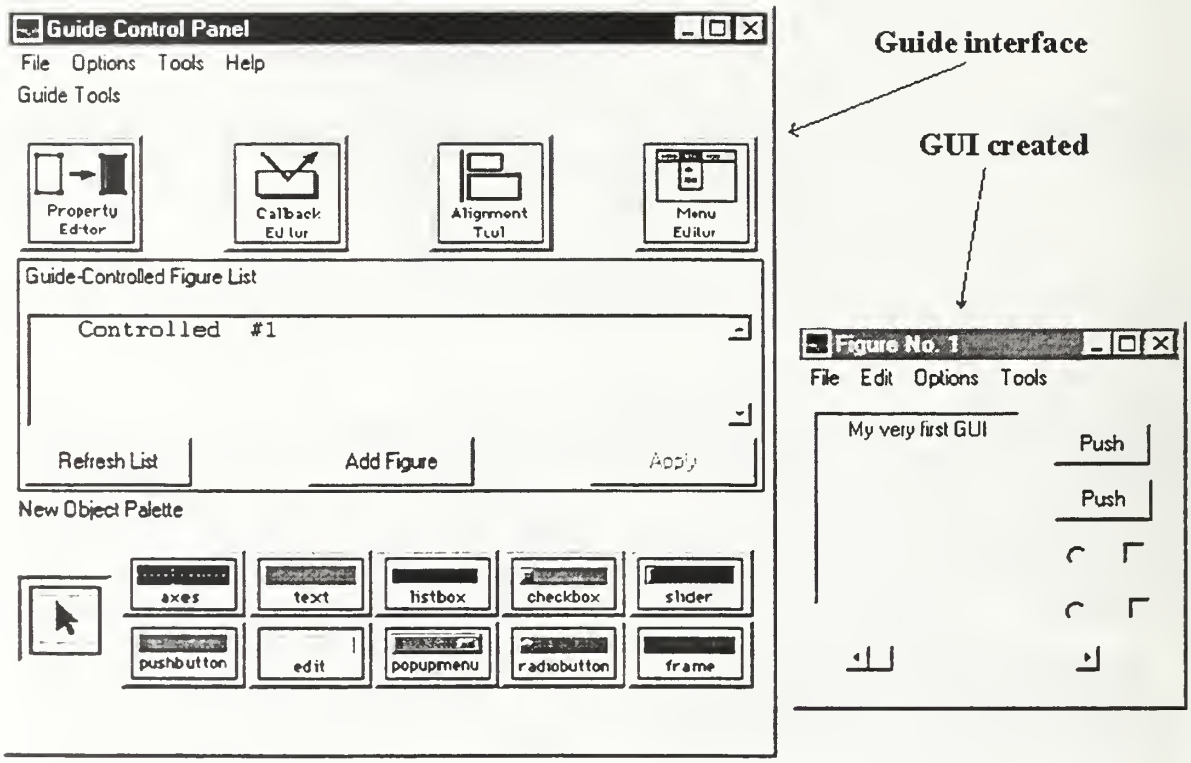

Figure 28. Matlab "From Ref. [18]"

APPENDIX B. SOURCE CODE

A. ROBOT PSDL-SERVER

Robot PSDL-server: file robot.psdl

```
TYPE a_data
 SPECIFICATION
END
IMPLEMENTATION ADA a_data
END

TYPE pv_data
 SPECIFICATION
END
IMPLEMENTATION ADA pv_data

END

TYPE pva_data
 SPECIFICATION
END
IMPLEMENTATION ADA pva_data

END

OPERATOR bop_calc
 SPECIFICATION
  INPUT
   bnss_pv_data : pv_data,
   bnss_state : pva_data
  OUTPUT
   bnss_a_data : a_data,
   bnss_error : INTEGER,
   bnss_pva_data : pva_data,
   bnss_state : pva_data
  MAXIMUM EXECUTION TIME 25 MS
END
IMPLEMENTATION ADA bop_calc

END

OPERATOR bop_display
 SPECIFICATION
  INPUT
   bnss_pva_data : pva_dat
  MAXIMUM EXECUTION TIME 5 MS
END

IMPLEMENTATION ADA bop_display
```

Robot PSDL-server: file robot.psdl (cont.)

```
END

OPERATOR bop_error
 SPECIFICATION
  INPUT
   bnss_error : INTEGER
  MAXIMUM EXECUTION TIME 5 MS
END
IMPLEMENTATION ADA bop_error

END

OPERATOR bop_fire_thrusters
 SPECIFICATION
  INPUT
   bnss_a_data : a_data
  MAXIMUM EXECUTION TIME 5 MS
END
IMPLEMENTATION ADA bop_fire_thrusters

END

OPERATOR bop_input
 SPECIFICATION
  OUTPUT
   bnss_pv_data : pv_data
  MAXIMUM EXECUTION TIME 5 MS
END
IMPLEMENTATION ADA bop_input

END

OPERATOR robot4
 SPECIFICATION
END
IMPLEMENTATION
 GRAPH
  VERTEX bop_calc : 25 MS

  VERTEX bop_display : 5 MS

  VERTEX bop_error : 5 MS

  VERTEX bop_fire_thrusters : 5 MS

  VERTEX bop_input : 5 MS

  EDGE bnss_a_data
   bop_calc ->
   bop_fire_thrusters
```

Robot PSDL-server: file robot.psdl (cont.)

```
  EDGE bnss_error
    bop_calc ->
    bop_error

  EDGE bnss_pv_data
    bop_input ->
    bop_calc

  EDGE bnss_pva_data
    bop_calc ->
    bop_display

  EDGE bnss_state
    bop_calc ->
    bop_calc
 DATA STREAM
  bnss_a_data : a_data,
  bnss_error : INTEGER,
  bnss_pv_data : pv_data,
  bnss_pva_data : pva_data,
  bnss_state : pva_data
 CONTROL CONSTRAINTS
  OPERATOR bop_calc
   PERIOD 50 MS

  OPERATOR bop_display
   TRIGGERED BY ALL
    bnss_pva_data

  OPERATOR bop_error
   TRIGGERED BY ALL
    bnss_error

  OPERATOR bop_fire_thrusters
   TRIGGERED BY ALL
    bnss_a_data

  OPERATOR bop_input
   PERIOD 200 MS
END
```

Robot PSDL-server: file ada.h

```
/*
-------------------------------------------------------------------------
-- FILE        : ada.h
-- CSCI        : Robot  Controller
-- Date        : June  1996
-- Author      :
-------------------------------------------------------------------------
*/


typedef struct {
      int               ix;
      int               iy;
      int               ir;
} Rec, *Rec_Ptr;
```

Robot PSDL-server: file global.h

```
/*
-------------------------------------------------------------------------
-- FILE        : global.h.a
-- CSCI        : Robot  Controller
-- Date        : June  1996
-- Author      :
-- Compiler    : Sun/Ada
-----------------------------------------------------------------
*/


struct data
{
int ix;
int iy;
int ir;
};
```

Robot PSDL-server: file bop_data.a

```
-------------------------------------------------------------------
-- FILE        : bop_a_data.a
-- CSCI        : Robot  Controller
-- Date        : June  1996
-- Author      :
-- Compiler    : Sun/Ada
-------------------------------------------------------------------


--
with text_io;
use text_io;

-- definition for a data stream of robot acceleration values

package a_data_PKG is

type a_data is record
        xacc : FLOAT := 0.0;
        yacc : FLOAT := 0.0;
end record;

end a_data_PKG;
```

Robot PSDL-server: file bop_calc.a

```
-------------------------------------------------------------------------
-- FILE       : bop_calc.a
-- CSCI       : Robot  Controller
-- Date       : June  1996
-- Author     :
-- Compiler   : Sun/Ada
-------------------------------------------------------------------------
with TEXT_IO;
use TEXT_IO;
with a_data_Pkg;
with pva_data_Pkg;
with pv_data_Pkg;
with math;
use math;

-- This is the central operator of the robot "soft landing process".
-- On the first call to this operator, initial input is prompted for.
-- On subsequent calls, this operator calculates the robot's distance
-- from the origin and adjusts acceleration as necessary to bring
-- the robot to a soft landing within the required "doughnut". The
-- algorithm used is to continually adjust thruster acceleration
-- as follows:
--
--        Calculate the robot's distance from origin.
--
--        If the robot is too close to the center,
--                issue an error message.
--        Else if inside the doughnut,
--                set acceleration to force robot to stop in 2 periods.
--                if the robot is moving very, very slow,
--                        issue a "DONE" message.
--        Else, the robot still has a ways to go, so
--                calculate thruster acceleration needed to reach origin
--                in 2 seconds (the robot may initially speed up, but it
--                should eventually experience continued slowdown as it
--                gets closer to the doughnut relative to it initial
--                distance from the doughnut).
--
--        If the robot is off the table,
--                issue an error message.
--
--        Calculate the distance moved since the last call to the this
--                operator based on this operator's calling period.
--
--        Calculate the robot's current velocity.
--
--        Write position, acceleration, velocity to outgoing data streams.
--
--        Save the operator's state.
```

Robot PSDL-server: file bop_calc.a (cont.)

```
--
--Spec
package bop_calc_Pkg is
        procedure bop_calc(bnss_state: in out pva_data_Pkg.pva_data; bnss_pva_data: out
pva_data_Pkg.pva_data; bnss_a_data: out a_data_Pkg.a_data; bnss_error: out INTEGER ;bnss_pv_data: in
pv_data_pkg.pv_data);
end bop_calc_Pkg;


--Body
package body bop_calc_Pkg is

  package FL_IO is new TEXT_IO.FLOAT_IO(FLOAT);

   firsttime                              : INTEGER := 1;

        procedure bop_calc(bnss_state: in out pva_data_Pkg.pva_data; bnss_pva_data: out
pva_data_Pkg.pva_data; bnss_a_data: out a_data_Pkg.a_data; bnss_error: out INTEGER ;bnss_pv_data: in
pv_data_Pkg.pv_data)
                is

                dis_from_center : FLOAT;
                xvel_desired: FLOAT;
                yvel_desired: FLOAT;
                tmpf : FLOAT;
   xpos                         : FLOAT := 0.0;
   ypos                      : FLOAT := 0.0;
   xvel                      : FLOAT := 0.0;
   yvel                      : FLOAT := 0.0;
   xacc                      : FLOAT := 0.0;
   yacc                      : FLOAT := 0.0;

 begin

        else -- load the previous state to local variables
         xpos := bnss_state.xpos;
         ypos := bnss_state.ypos;
         xvel := bnss_state.xvel;

         yvel := bnss_state.yvel;
         xacc := bnss_state.xacc;
         yacc := bnss_state.yacc;
        end if;

        if bnss_pv_data.irequest /=  0 then
          xpos := FLOAT(bnss_pv_data.xpos)/1000.0;
          ypos := FLOAT(bnss_pv_data.ypos)/1000.0;
       end if;
        bnss_error  := 0;
```

Robot PSDL-server: file bop_calc.a

```
    -- Calculate Distance from Center
dis_from_center := Sqrt(xpos * xpos + ypos *ypos);

if dis_from_center <  0.01  then  -- In to close to center
            PUT_LINE("Too Close");
            bnss_error := 1;
    elsif dis_from_center < 0.02 then -- Inside circle
            xacc := -xvel * 20.0; -- Stop in 2 periods
     yacc := -yvel * 20.0;

     if abs(yvel) < 0.00001 then
            if abs(xvel) < 0.00001 then
                PUT("DONE");
            end if;
            end if;
else
            -- Adjust acceleration
     xvel_desired := - ( xpos / 2.0); -- Reach center in 2s
     yvel_desired := - ( ypos / 2.0);
     xacc := - ( xvel - xvel_desired);
     yacc := - ( yvel - yvel_desired);
     end if;

    if abs(xpos) > 1.0 then -- Off the table
    PUT_LINE("OFF THE TABLE X");
            bnss_error := 2;
end if;
if abs(ypos) > 1.0 then
    PUT_LINE("OFF THE TABLE Y");
            bnss_error := 3;
end if;



    -- Calculate Distance Moved
xpos := xpos + (xvel * 0.05) + (xacc * 0.00125);
ypos := ypos + (yvel * 0.05) + (yacc * 0.00125);
    -- Calculate New Velocity

xvel := xvel + (xacc * 0.05);
yvel := yvel + (yacc * 0.05);

    -- write to output data streams
    bnss_pva_data.xpos := xpos;
    bnss_pva_data.ypos := ypos;
    bnss_pva_data.xvel := xvel;
    bnss_pva_data.yvel := yvel;
    bnss_pva_data.xacc := xacc;
    bnss_pva_data.yacc := yacc;
    bnss_a_data.xacc := xacc;
    bnss_a_data.yacc := yacc;
```

Robot PSDL-server: file bop_calc.a (cont.)

```
        -- save this operators state
        bnss_state.xpos := xpos;
        bnss_state.ypos := ypos;
        bnss_state.xvel := xvel;
        bnss_state.yvel := yvel;
        bnss_state.xacc := xacc;
        bnss_state.yacc := yacc;

  end bop_calc;
end bop_calc_Pkg;
```

Robot PSDL-server: file bop_display.a

```
-----------------------------------------------------------------------
-- FILE        : bop_display.a
-- CSCI        : Robot  Controller
-- Date        : June  1996
-- Author      :
-- Compiler    : Sun/Ada
-----------------------------------------------------------------------

with TEXT_IO;
use TEXT_IO;
with pva_data_Pkg;
with socket_Pkg;
use socket_Pkg;


package bop_display_Pkg is
        procedure bop_display(bnss_pva_data: in pva_data_Pkg.pva_data);
end bop_display_Pkg;


package body bop_display_Pkg is

  package FL_IO is new TEXT_IO.FLOAT_IO(FLOAT);

        procedure bop_display(bnss_pva_data: in pva_data_Pkg.pva_data) is

     ix: INTEGER;
     fx: FLOAT;
     fy: FLOAT;


  begin

     ix := 0;
     fx := bnss_pva_data.xpos;
     fy := bnss_pva_data.ypos;
     socket(fx,fy,ix);

  end bop_display;
end bop_display_Pkg;
```

Robot PSDL-server: file bop_error.a

```
------------------------------------------------------------------------
-- FILE        : bop_error.a
-- CSCI        : Robot  Controller
-- Date        : June  1996
-- Author      :
-- Compiler    : Sun/Ada
------------------------------------------------------------------------


with TEXT_IO;
use TEXT_IO;


-- All that this operator does is read an error flag
-- setting from its input data stream and writes
-- and error indication to the display if the error
-- flag is set to indicate error.

--Spec
package bop_error_Pkg is
          procedure bop_error(bnss_error: in INTEGER);
end bop_error_Pkg;


--Body
package body bop_error_Pkg is

  package INT_IO is new TEXT_IO.INTEGER_IO(INTEGER);

  procedure bop_error(bnss_error: in INTEGER) is

  begin
                    if bnss_error > 0 then
                    INT_IO.PUT(bnss_error);
                    PUT_LINE("Error");
                    end if;
  end bop_error;
end bop_error_Pkg;
```

Robot PSDL-server: file bop_fire_thrusters.a

```
-------------------------------------------------------------------
-- FILE        : bop_fire_thrusters.a
-- CSCI        : Robot  Controller
-- Date        : June  1996
-- Author      :
-- Compiler    : Sun/Ada
-------------------------------------------------------------------


with TEXT_IO;
use TEXT_IO;
with a_data_Pkg;

-- This is sort of a dummy operator that inputs thruster acceleration
-- adjustments and outputs the received adjustment to the system
-- display. Its intended purpose was to effect a more realistic
-- model by having an operator that actually receives the thruster
-- adjustments from the operator that does the thruster adjustment
-- calculations.

package bop_fire_thrusters_Pkg is
        procedure bop_fire_thrusters(bnss_a_data: in a_data_Pkg.a_data);
end bop_fire_thrusters_Pkg;


package body bop_fire_thrusters_Pkg is

  package FL_IO is new TEXT_IO.FLOAT_IO(FLOAT);

        procedure bop_fire_thrusters(bnss_a_data: in a_data_Pkg.a_data) is

        ftmp : FLOAT := 0.0;

 begin

        if bnss_a_data.xacc > 0.0 then
          Put(" Left Thruster =");
          FL_IO.PUT(bnss_a_data.xacc,0,3,0);
          Put(" Right Thruster =");
          FL_IO.PUT(ftmp,0,3,0);
        else
          Put(" Left Thruster =");
          FL_IO.PUT(ftmp,0,3,0);
          Put(" Right Thruster =");
          FL_IO.PUT(abs(bnss_a_data.xacc),0,3,0);
        end if;
```

Robot PSDL-server: file bop_fire_thrusters.a (cont.)

```
        if bnss_a_data.yacc > 0.0 then
          Put(" Bott Thruster =");
          FL_IO.PUT(bnss_a_data.yacc,0,3,0);

          Put(" Top  Thruster =");
          FL_IO.PUT(ftmp,0,3,0);
        else
          Put(" Bott Thruster =");
          FL_IO.PUT(ftmp,0,3,0);
          Put(" Top  Thruster =");
          FL_IO.PUT(abs(bnss_a_data.yacc),0,3,0);
        end if;
        NEW_LINE;

  end bop_fire_thrusters;
end bop_fire_thrusters_Pkg;
```

Robot PSDL-server: file bop_fire_inputs.a

```
-------------------------------------------------------------------------
-- FILE        : bop_input.a
-- CSCI        : Robot  Controller
-- Date        : June  1996
-- Author      :
-- Compiler    : Sun/Ada
-------------------------------------------------------------------------

with TEXT_IO;
use TEXT_IO;
with socket_Pkg;
use socket_Pkg;
with pv_data_Pkg;

package body bop_input_Pkg is

      procedure bop_input(bnss_pv_data: out pv_data_Pkg.pv_data) is

      ir: INTEGER;
      fx: FLOAT;
      fy: FLOAT;

 begin

      ir := 1;
      fx := 0.0;
      fy := 0.0;
      socket(fx,fy,ir);          -- Check the GUI for input
           bnss_pv_data.xpos := fx;
           bnss_pv_data.ypos := fy;
           bnss_pv_data.irequest := ir;

  end bop_input;
end bop_input_Pkg;
```

Robot PSDL-server: file bop_pv_data.a

```
-------------------------------------------------------------------
-- FILE        : bop_pv_data.a
-- CSCI        : Robot  Controller
-- Date        : June  1997
-- Author      :
-- Compiler    : Sun/Ada
-------------------------------------------------------------------


--
with text_io;
use text_io;

-- definition for a data stream of robot position,
-- velocity, and acceleration values

package pv_data_PKG is

type pv_data is record
        irequest : INTEGER := 0;
        xpos : FLOAT := 0.0;
        ypos : FLOAT := 0.0;
        xvel : FLOAT := 0.0;
        yvel : FLOAT := 0.0;
end record;

end pv_data_PKG;
```

Robot PSDL-server: file bop_pva_data.a

```
-----------------------------------------------------------------------
-- FILE        : bop_pva_data.a
-- CSCI        : Robot  Controller
-- Date        : June  1996
-- Author      :
-- Compiler    : Sun/Ada
-----------------------------------------------------------------------


--
with text_io;
use text_io;

-- definition for a data stream of robot position,
-- velocity, and acceleration values

package pva_data_PKG is

type pva_data is record
        xpos : FLOAT := 0.0;
        ypos : FLOAT := 0.0;
        xvel : FLOAT := 0.0;
        yvel : FLOAT := 0.0;
        xacc : FLOAT := 0.0;
        yacc : FLOAT := 0.0;
end record;

end pva_data_PKG;
```

Robot PSDL-server: file socket.a

```
-----------------------------------------------------------------------
-- FILE        : socket.a
-- CSCI        : Robot  Controller
-- Date        : June  1996
-- Author      :
-- Compiler    : Sun/Ada

--Purpose    :  The package contains the procedure socket
--which is used to pass data to GUI-client.
--Socket binds to the C program that does the
--actual socket connection.
-----------------------------------------------------------------------

with text_io;
use text_io;

--Spec
package socket_PKG is
      procedure socket(xx: in out Float; yy: in out Float;
                       ireq: in out INTEGER);
end socket_PKG;



--pkg
package body socket_PKG is

 -- to instantiate integer generic

                       a1 : FLOAT;
                       b1 : FLOAT;

          type Rec is record
                       ix: Integer ;
                       iy: Integer ;
                       ir: Integer ;
          end record;

          type Rec_Ptr is access Rec;

          l : Rec_Ptr := new Rec;

      pragma LINK_WITH ("server.o");
      procedure server(l : in Rec_Ptr);
      pragma Interface (C, server );

      procedure Socket(xx: in out Float; yy: in out Float;
                       ireq: in out INTEGER) is
```

Robot PSDL-server: file socket.a (cont.)

```
    begin

                a1 := xx * 1000.0;
                I.ix := INTEGER(a1);
                b1 := yy * 1000.0;
                I.iy := INTEGER(b1);
                I.ir := INTEGER(ireq);
                server(I);
                xx := FLOAT(I.ix);
                yy := FLOAT(I.iy);
                ireq := I.ir;

      end Socket;


end socket_PKG;
```

Robot PSDL-server: file server.c

```
/*
-------------------------------------------------------------------------
-- FILE        : server.c
-- CSCI        : Robot  Controller
-- Date        : June  1997
-- Compiler    : gcc
-------------------------------------------------------------------------

-- This program implements the C socket connection.
*/

#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <signal.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/filio.h>
#include "confer.h"
#include "global.h"
#include "parse.h"
#include "ada.h"

#define MAX 128
#define MAXFILTERS 100
#define MAXRECORDS 100

extern int errno;

void close_connection();

struct sockaddr_in My_Sockaddr;
int isock;
struct data recmessage;
struct data sendmessage;
```

Robot PSDL-server: file server.c (cont.)

```
/*****************************************
 Function  server
 Arguments: Rec_Ptr Info;
 Return:  None
 Purpose: Sends data to the client.
      The struct defined in message being the structure
      data defined in global.h.  Will return 1 if able send.
*****************************************/
server(Info)
 Rec_Ptr Info;
{
 int ret;

  sendmessage.ix = Info->ix ;
  sendmessage.iy = Info->iy ;
  sendmessage.ir = Info->ir ;

  ret = inet_server();
  if(ret < 0)
         printf("failed to read message socket may have died\n");

  Info->ix = recmessage.ix;
  Info->iy = recmessage.iy;
  Info->ir = recmessage.ir;
}

/*********************************************
 Function  inet_server
 Arguments: None
 Return: integer
          -1 if fails to read or socket died
 Purpose: Sets up server. Will return 1 if able
          to read.  Will return -1 if fails or socket has
          died.

*********************************************/
inet_server()
{
 int result;

   signal(SIGPIPE, close_connection);
```

Robot PSDL-server: file server.c (cont.)

```
  if(isock < 1)
        {
        set_up();
        }
  if(isock < 1)
        {
        return(-1);
        }
        result = msgs();

        /* Determine whether we've exited because of an error or the
         * connection has been disconnected.
         */
        if (result < 0)
    {
            perror("read failed");
                isock = 0;
            return(-1);
        }
    if(result < 1)  /* socket died*/
        {
                isock = 0;
                return(-1);
        }
        return(1);
}
/*******************************************************
 Function: setup
 Arguments: none
 Return: int (-1 if fails)
 Purpose: Continues the socket setup.  Listens for a connection.
*******************************************************/
int set_up()
{
  static struct sockaddr_in from;
  static int length = sizeof(from);
  static int firsttime = 1;

  if (firsttime)
  {
   /* Set-up the connection  */
   if(init_isock_conn() < 0)
   {
        return(-1);
   };
   /* Get ready to accept from */
   if(listen(LISTENER, 1) < 0) {
        perror("listen");
        return(-1);
   }
  firsttime = 0;
```

Robot PSDL-server: file server.c (cont.)

```
        /* Accept new data connection */
        if((isock = accept(LISTENER, &from, &length)) < 0)
           Error_Abort("accept");

        fprintf(stderr, "connected\n");
 }

/*****************************************************
 Function: msgs
 Arguments: none
 Return: int
 Purpose: Performs the reading and writing on the socket.
*****************************************************/
msgs()
{
   static int ii = 0;
   int rest;
   struct sockaddr_in from;
   int from_len = sizeof(struct sockaddr_in);
   int remnant = 0, cc;
   int itmp,ret,bytes;
   static int count = 1;
   char test[100];
   int icount = 0;
   ret=0;

        write(isock, &sendmessage, sizeof(sendmessage));
         if(isock > 0)
          cc = read(isock, test,1);    /* read check byte */
         if(test[0] == 1)
         {
          if(isock > 0)
           cc = read(isock, &recmessage,sizeof(recmessage)); /* read data structure */
         }
         else
         {
                recmessage.ix = 0;
                recmessage.iy = 0;
                recmessage.ir = 0;
                return(1);
         }
         rest = 12-cc;
         if(rest != 0)
         {
          cc = recvfrom(isock, test,rest,0,&from,&from_len);
          rest = rest - cc;
         }
        return(1);
}
```

Robot PSDL-server: file server.c (cont.)

```
/*****************************************************
 Function: close_connection
 Arguments: none
 Return: void
 Purpose:  Close the socket when a SIGPIPE accurs.
*****************************************************/
void close_connection()
{
   shutdown(isock, 2);
   fprintf(stderr, "SIGPIPE received...exiting\n");
   close(isock);
   isock = 0;

}
/*
 * Set-up a TCP connection
 */
int init_isock_conn()
{
   int sock;

   struct sockaddr_in server;
   struct hostent *hp;
   struct servent *sp;

   bzero((char *)&server, sizeof(server));

   if((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
   {
         Error_Abort("socket");
         return(-1);
   }
   if(setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, (char *)0,0) < 0)
         Error_Abort("setsockopt,1");

   if(setsockopt(sock, SOL_SOCKET, SO_DONTLINGER, (char *)0,0) < 0)
         Error_Abort("setsockopt,2");
/*
   if((sp = getservbyname(service, "tcp")) == NULL) {
         fprintf(stderr, "isock: tcp/isock service not available\n");
         exit(-1);
   }
   server.sin_port = sp->s_port;
*/
   server.sin_port = 4141;  /* Hard code port number , should use services file */

   server.sin_family = AF_INET;
   server.sin_addr.s_addr = INADDR_ANY;
```

Robot PSDL-server: file server.c (cont.)

```
if(bind(sock, &server, sizeof(server)) < 0)
   {
          perror("bind");
          close(sock);
          return(-1);
   }
if (sock != LISTENER) {
          if (dup2(sock, LISTENER) == -1)
        fprintf(stderr, "\nINIT_CONN: dup2 failed\n");


   }
   return(1);
}
```

B. ROBOT JAVA GUI-CLIENT

Robot JAVA GUI-client : file lander.html

JAVA - Applet GUI-Client

```
<html>
<head>
<title>CAPS Robot Lander Craft - Client/Server JAVA JDK
 </title>
</head>
<body background="ste.jpg" bgcolor="#FFFFFF">
<h2>
NRAD/NPS MS Software Engineering --
</h2>
<p>
CAPS -  Client/Server JAVA -  Robot Lander
<p>
<APPLET CODE="Lander" WIDTH=400 HEIGHT=500>
</APPLET>
</HTML>
</body>
```

Robot JAVA GUI-client: file lander.java

```
/*********************************************************
 * File:  lander.java
 *
 *    Lander is an applet GUI-client for the robot.  Communications with
 *    the PSDL-server is via socket communications.  The x,y locations are
 *    passed from the server.
 *
 * Date:  8/97
 *
 ********************************************************* */

import java.awt.*;
import java.applet.*;
import java.util.*;
import java.net.*;
import java.lang.*;
import java.io.*;

public class Lander extends Applet implements Runnable
{
        MediaTracker theMediaTracker;
        Image imageDuke,imageBack;
        Image imageBuffer;
        Image buffer;
        Graphics imageBufferG, bufferG;
        Thread t1;
        //RandClient r1;
        int xx,yy;
        int newx,newy;
        double dcompress = 1.0;
        boolean bNeedToRestart = false;
        boolean bRunning = false;

        TextField xField,yField;
        TextField xvField,yvField;
        TextField xaField,yaField;
        Button startButton;
        Button bZoomIn,bZoomOut;

        public void start()
        {
                System.out.println("start ");
                xx = 0;
                yy = 0;

        }
```

Robot JAVA GUI-client: file lander.java (cont.)

```
        public boolean action(Event evt,Object arg)
        {
                boolean retVal = false;

                if(evt.target == bZoomIn)
                {
                        dcompress = dcompress*1.2;
                        buildback();
                        repaint();
                        retVal = true;
                }
                if(evt.target == bZoomOut)
                {
                        dcompress = dcompress/1.2;
                        buildback();
                        repaint();
                        retVal = true;
                }
                if((evt.target == startButton)
                        ||(evt.target == xField)
                        ||(evt.target == yField))
                {
                        String xvalue,yvalue;
                        double x,y;

                        if(bRunning == false)
                        {
                         bRunning = true;
                         t1 = new Thread(this);
                      t1.start();
                         repaint();
                        }

                        xvalue = xField.getText();
                        yvalue = yField.getText();

                        if((xvalue != null) && (yvalue != null) && !xvalue.equals("") &&
!yvalue.equals(""))
                        {
                                //cvalue = Double.toString(c);
                                //cField.setText(cvalue);
                                newx = Integer.parseInt(xvalue);
                                newy = Integer.parseInt(yvalue);
                                bNeedToRestart = true;
                                xx = newx;
                                yy = newy;
```

Robot JAVA GUI-client: file lander.java (cont.)

```
                        repaint();
                    }
                    retVal = true;
                }
                return retVal;
        }

        public void update(Graphics g)
        {

                paint(g);
        }

        public void paint(Graphics g)
        {
                Rectangle bounds;
                int w,h;
                double dscreenx, dscreeny;
                Color c;
                int i,max;

                bounds = this.bounds();

                dscreenx = (double)xx*dcompress/10 + 200;
                dscreeny = (double)yy*dcompress/10 + 200;

            bufferG.drawImage(imageBuffer, 0, 0, this);
                w = imageDuke.getWidth(this);
                h = imageDuke.getHeight(this);
                bufferG.drawImage(imageDuke,(int)dscreenx-w/2,(int)dscreeny-h/2,w,h,this);
                g.drawImage(buffer, 0, 0, this);


         }
        public boolean mouseDrag(Event event, int screenx, int screeny)
        {
                newx = (int)((  (double)((10 * screenx) - 2000)  )/dcompress);
                newy = (int)((  (double)((10 * screeny) - 2000)  )/dcompress);
                xx = newx;
                yy = newy;
                bNeedToRestart = true;
                repaint();
                return true;
        }

        public void stop()
        {
                System.out.println("stop ");
                bRunning = false;
        }
```

```
public void init()
{
        Rectangle bounds;
        Panel toolbar,toolbar2;

        bounds = this.bounds();
        System.out.println("init ");

        this.setLayout(new GridLayout(14,1));
        toolbar = new Panel();
        toolbar2 = new Panel();
        xField = new TextField("1000",6);
        yField = new TextField("1000",6);
        xvField = new TextField(6);
        yvField = new TextField(6);
        xaField = new TextField(6);
        yaField = new TextField(6);

        startButton = new Button("Restart");
        toolbar.add(startButton);
        bZoomIn = new Button("Zoom In");
        toolbar.add(bZoomIn);
        bZoomOut = new Button("Zoom Out");
        toolbar.add(bZoomOut);

        toolbar.add(new Label("XY",Label.RIGHT));
        toolbar.add(xField);
        toolbar.add(yField);

        this.add(toolbar);
        toolbar2.add(xvField);
        toolbar2.add(yvField);
        toolbar2.add(new Label("XYacc",Label.RIGHT));

        toolbar2.add(xaField);
        toolbar2.add(yaField);
        this.add(toolbar2);

        //"dukes.gif");
        //"photo_7.gif");
        imageDuke = getImage(getDocumentBase(), "dukes.gif");
        // ent.gif
        //imageBack = getImage(getDocumentBase(), "sat.gif");

        toolbar2.add(new Label("XYvel",Label.RIGHT));
        imageBuffer = createImage(size().width, size().height);
```

Robot JAVA GUI-client: file lander.java (cont.)

```
                imageBufferG = imageBuffer.getGraphics();

                buffer = createImage(size().width, size().height);
                bufferG = buffer.getGraphics();


                theMediaTracker = new MediaTracker(this);
                //theMediaTracker.addImage(imageBack,0);
                theMediaTracker.addImage(imageDuke,1);
                try
                {
                        theMediaTracker.waitForAll(2000);
                }
                catch(InterruptedException ex) { }

                buildback();
        }

        public void buildback()
        {
                double dx,dtmp,dtmp2;
                Rectangle bounds;
                int w,h,ii;

                bounds = this.bounds();
                //System.out.println("build "+bounds.width);

                //imageBufferG.clearRect(0,0,bounds.width,bounds.height);
                imageBufferG.setColor(Color.black);
                imageBufferG.fillRect(0,0,bounds.width,bounds.height);

                //w = imageBack.getWidth(this);
                //h = imageBack.getHeight(this);
                //imageBufferG.drawImage(imageBack,200-(int)((double)w*dcompress)/(2*4),200-
(int)((double)h*dcompress)/(2*4),(int)((double)w*dcompress/4.0),(int)((double)h*dcompress/4.0),this);

                dtmp = dcompress * 100.0;
                dtmp2 = dcompress * 200.0;
                imageBufferG.setColor(Color.yellow);
                imageBufferG.drawOval(200-(int)dtmp,200-(int)dtmp,(int)dtmp2,(int)dtmp2);
                dtmp = dcompress * 2.0;
                dtmp2 = dcompress * 4.0;
                imageBufferG.setColor(Color.blue);
                imageBufferG.drawOval(200-(int)dtmp,200-(int)dtmp,(int)dtmp2,(int)dtmp2);
                dtmp = dcompress * 1.0;
                dtmp2 = dcompress * 2.0;
                imageBufferG.setColor(Color.cyan);
                imageBufferG.drawOval(200-(int)dtmp,200-(int)dtmp,(int)dtmp2,(int)dtmp2);
```

```
            imageBufferG.setColor(Color.gray);
            imageBufferG.drawLine(200,0,200,400);
            imageBufferG.drawLine(0,200,400,200);

            dx = 0.0;
            ii = 0;
            while(dx < 200.0)
            {
             dx = (double)ii * dcompress;
             imageBufferG.drawLine(200 + (int)dx,196,200 + (int)dx,204);
             imageBufferG.drawLine(200 - (int)dx,196,200 - (int)dx,204);
             imageBufferG.drawLine(196,200 + (int)dx,204,200 + (int)dx);
             imageBufferG.drawLine(196,200 - (int)dx,204,200 - (int)dx);
             if(dcompress < 2.0)
              ii = ii + 10;
             else
              ii = ii + 1;
            }
}
public void destroy()
{
            System.out.println("destroy ");
}
public void run()
{
            Socket server;
            InputStream in;
            OutputStream out;
            DataInputStream dataIn;
            DataOutputStream dataOut;
            int ii = 1;
            int iin;

            try
            {
             t1.sleep(1000);
            }
            catch(InterruptedException ex) { }


   while(bRunning)
   {

            try
            {
                        //server = new Socket(InetAddress.getLocalHost(),4141);
                        server = new Socket("pegasi.nosc.mil",4141);
                        in = server.getInputStream();
                        out = server.getOutputStream();
```

Robot JAVA GUI-client: file lander.java (cont.)

```
                    dataIn = new DataInputStream(in);
                    dataOut = new DataOutputStream(out);

                    while(bRunning)
                    {

                    iin = dataIn.readInt();
                    //System.out.println("The current number is: "+iin);
                    xx = iin;
                    iin = dataIn.readInt();
                    //System.out.println("The current number is: "+iin);
                    yy = iin;
                    iin = dataIn.readInt();
                    //System.out.println("The current number is: "+iin);
                    if(iin == 1)
                    {
                     if(bNeedToRestart)

                     {
                     dataOut.writeByte(1);
                     dataOut.writeInt(newx);
                     dataOut.writeInt(newy);
                     dataOut.writeInt(1);
                     //System.out.println("The x: "+newx);
                     //System.out.println("The y: "+newy);
                     bNeedToRestart = false;
                     }
                     else
                     {
                     dataOut.writeByte(65);
                     }
                    }
                    else
                    {
                     dataOut.writeByte(65);
                     repaint();
                    }
                    ii = ii + 1;
                    }

                    System.out.println("Closing socket");
                    dataIn.close();
                    in.close();
                    dataOut.close();
                    out.close();
                }
                catch(IOException ex)
                {
//                  System.out.println("Error connecting to random server");
                }
        }
```

Robot JAVA GUI-client: file lander.java (cont.)

```
}

        public boolean imageUpdate(Image img, int flags, int x, int y, int w, int h)
        {
                if(img != imageDuke)
                {
                buildback();
                repaint();
                return true;
                }
                repaint();
                return true;
        }
}
```

C. ROBOT VISUAL WORKSHOP (MOTIF) GUI-CLIENT

Robot_Visual Workshop (Motif) GUI-client: file: Makfile

```
SHELL=/bin/sh
UILFLAGS=-I/usr/include/uil
MRMLIBS=-lMrm

# solaris 2.x
XINCLUDES=-I/usr/dt/include -I/usr/openwin/include -I/usr/openwin/include/X11
XLIBS=-L/usr/dt/lib -L/usr/openwin/lib -R/usr/dt/lib -R/usr/openwin/lib
LDLIBS=-lgen -lthread -lsocket -lnsl -lm
CCC=CC
MRMLIBS=-L/usr/dt/lib -lMrm
UILFLAGS=-I/usr/include/uil -I/usr/dt/include/uil
VISUROOT=/opt/SUNWspro/WS4.0

XPMLIBDIR = ${VISUROOT}/user_widgets/obj
XPMDIR = ${VISUROOT}/contrib/xpm/lib
LDFLAGS = ${XLIBS} -L${XPMLIBDIR}
MOTIFLIBS = -lXpm -lXm -lXt -lX11

XPCLASS = $(VISUROOT)/src/xdclass
XPCLASSLIBS = $(XPCLASS)/lib/libxdclass.a

CFLAGS=-I. ${XINCLUDES} -I${XPMDIR}
CCFLAGS=${CFLAGS} -I${XPCLASS}/h

UIL=uil

#DO NOT EDIT >>>
XD_C_PROGRAMS=\
                bp

#<<< DO NOT EDIT

#DO NOT EDIT >>>
XD_C_PROGRAM_OBJECTS=\
                bp.o
#<<< DO NOT EDIT

#DO NOT EDIT >>>
XD_C_PROGRAM_SOURCES=\
                bp.c

#<<< DO NOT EDIT

#DO NOT EDIT >>>
XD_C_STUB_OBJECTS=\
                bp_stubs.o
```

Robot_Visual Workshop (Motif) GUI-client: file: Makefile (cont.)

```
#<<< DO NOT EDIT

#DO NOT EDIT >>>
XD_C_STUB_SOURCES=\
                bp_stubs.c

#<<< DO NOT EDIT

XD_ALL_C_SOURCES=$(XD_C_PROGRAM_SOURCES) $(XD_C_SOURCES)
$(XD_C_STUB_SOURCES) $(XD_C_FOR_UIL_PROGRAM_SOURCES)
$(XD_C_FOR_UIL_SOURCES)

XD_ALL_CC_SOURCES=$(XD_CC_PROGRAM_SOURCES) $(XD_CC_SOURCES)
$(XD_CC_STUB_SOURCES)

all: CHECKENV CHECKROOT $(XD_C_PROGRAMS) $(XD_CC_PROGRAMS)
$(XD_C_FOR_UIL_PROGRAMS) $(XD_UIL_OBJECTS)

depend:
        makedepend -- $(CFLAGS) $(CPPFLAGS) -- $(XD_ALL_C_SOURCES)
        makedepend -a -- $(CCFLAGS) $(CPPFLAGS) -- $(XD_ALL_CC_SOURCES)

clean:
        rm -f $(XD_C_PROGRAMS) $(XD_C_PROGRAM_OBJECTS) $(XD_C_OBJECTS) \
            $(XD_CC_PROGRAMS) $(XD_CC_PROGRAM_OBJECTS) $(XD_CC_OBJECTS) \
            $(XD_C_STUB_OBJECTS) \
            $(XD_CC_STUB_OBJECTS) \
            $(XD_C_FOR_UIL_PROGRAMS) $(XD_C_FOR_UIL_PROGRAM_OBJECTS)
$(XD_C_FOR_UIL_OBJECTS) \
            $(XD_UIL_OBJECTS)

CHECKENV:
        @test -n "$(VISUROOT)" || (echo You must set \$$VISUROOT in the makefile or in your shell
environment; exit 1)

CHECKROOT:
        @test -d "$(XPCLASS)" || (echo \$$VISUROOT must point to a valid root directory; exit 1)
        @test -d "$(VISUROOT)/make_templates" || (echo \$$VISUROOT must point to a valid root
directory; exit 1)

#DO NOT EDIT >>>
bp: client.o th1.o lander.o bp.o $(XD_C_OBJECTS) $(XD_C_STUB_OBJECTS)
        $(CC) -g $(CFLAGS) $(CPPFLAGS) $(LDFLAGS) -o bp client.o th1.o lander.o bp.o
$(XD_C_OBJECTS) $(XD_C_STUB_OBJECTS) $(MOTIFLIBS) $(LDLIBS)
#<<< DO NOT EDIT

#DO NOT EDIT >>>
bp.o: bp.c
        $(CC) -g $(CFLAGS) $(CPPFLAGS) -c bp.c
```

Robot_Visual_Workshop (Motif) GUI-client: _file: Makefile (cont.)

```
#<<< DO NOT EDIT

#DO NOT EDIT >>>
bp_stubs.o: bp_stubs.c
	$(CC) -g $(CFLAGS) $(CPPFLAGS) -c bp_stubs.c
#<<< DO NOT EDIT
```

```
/*
** Generated by WorkShop Visual
*/

/*
** WorkShop Visual-generated prelude.
** Do not edit lines before "End of WorkShop Visual generated prelude"
** Lines beginning ** WorkShop Visual Stub indicate a stub
** which will not be output on re-generation
*/

/*
**LIBS: -lXm -lXt -lX11
*/

#include <X11/Xatom.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>

#include <Xm/Xm.h>
#include <Xm/DrawingA.h>
#include <Xm/PushB.h>

#include "bp.h"


extern void XDmanage_link ( Widget w, XtPointer client_data, XtPointer call_data );
extern void XDunmanage_link ( Widget w, XtPointer client_data, XtPointer call_data );
extern void XDpopup_link ( Widget w, XtPointer client_data, XtPointer call_data );
extern void XDpopdown_link ( Widget w, XtPointer client_data, XtPointer call_data );
extern void XDmap_link ( Widget w, XtPointer client_data, XtPointer call_data );
extern void XDunmap_link ( Widget w, XtPointer client_data, XtPointer call_data );
extern void XDenable_link ( Widget w, XtPointer client_data, XtPointer call_data );
extern void XDdisable_link ( Widget w, XtPointer client_data, XtPointer call_data );

/* End of WorkShop Visual generated prelude */

/*
** WorkShop Visual Stub mybrucecallback
*/

void mybrucecallback(Widget w, XtPointer client_data, XtPointer xt_call_data)
{
        static int firsttime = 1;

        XmPushButtonCallbackStruct *call_data = (XmPushButtonCallbackStruct *) xt_call_data ;
```

```
        printf("B1\n");

        if(firsttime)
        {
         oldmain();
         firsttime = 0;
        }
        send_data(1000,1000);
}

/*
** WorkShop Visual Stub bruce_drawing_callback_resize
*/

void bruce_drawing_callback_resize(Widget w, XtPointer client_data, XtPointer xt_call_data)
{
        GC gc;
        static int firsttime = 1;

        XmDrawingAreaCallbackStruct *call_data = (XmDrawingAreaCallbackStruct *) xt_call_data ;
        printf("B2\n");

        if(!firsttime)
        {
         setup_image(w);
         drawback();
        }
        firsttime = 0;


}

/*
** WorkShop Visual Stub bruce_drawing_callback_input
*/

void bruce_drawing_callback_input(Widget w, XtPointer client_data, XtPointer xt_call_data)
{
        GC gc;

        XmDrawingAreaCallbackStruct *call_data = (XmDrawingAreaCallbackStruct *) xt_call_data ;
        printf("B3\n");

}
```

```
! Generated by WorkShop Visual


! bp_shell

! messageBox1

! bruce_button
XApplication*bruce_button.labelString: Restart

! bruce_drawing
XApplication*bruce_drawing.foreground: SlateGrey
XApplication*bruce_drawing.background: GhostWhite
XApplication*bruce_drawing.marginWidth: 400
XApplication*bruce_drawing.marginHeight: 400

! cascade1
XApplication*cascade1.labelString: File

! button1
XApplication*button1.labelString: Open

! button4
XApplication*button4.labelString: Exit

! cascade2
XApplication*cascade2.labelString: Edit
```

```
module 'XApplication'
version = 43;
applicationName = 'XApplication';
generateNameC = * 'bp.c';
generateNameCMainProgram = * 'bp.c';
generateNameStubs = * 'bp_stubs.c';
generateNameExterns = * 'bp.h';
generateNameResDB = * 'bp.res';
generateNameCPixmaps = * 'bp_pixmaps.h';
generateCHeaderFile = 'bp.h';
generateNameMakefile = 'Makefile';
generateMask = 12;
useMask = 1;
ansiC = 1;
generateNewMakefile = 1;
generateMakeTemplate = 0;
useCPixmaps = 0;
useUILPixmaps = 0;
useCHeaders = 1;
useCUILHeaders = 0;
CPPFlavour = 0;
useCPPHeaders = 0;
useCPPHeadersMFCWindows = 0;
useCPPHeadersMFCMotif = 0;
object 'bp_shell' : XmDialogShell {
        arguments {
        lastGenName = 'bp_shell';
        createPreludeStatus = 2;
        preInstantiation = 'void create_bp_shell (Display *display, char *app_name, int app_argc, char
**app_argv)
';
        XmNallowShellResize = true;
        XmNprimary = 1;
        };
object '1' : XmMessageTemplate {
        arguments {
        lastGenName = 'messageBox1';
        XmNautoUnmanage = false;
        XmNdialogType = 0;
        };
object '1' : XmSeparator GADGET {
        arguments {
        name = 'Separator';
        lastGenName = 'separator1';
        };
};
object 'bruce_button' : XmPushButton {
        arguments {
        lastGenName = 'bruce_button';
```

```
            XmNlabelString = 'Restart';
            };
            callbacks {
            XmNactivateCallback = '
mybrucecallback( )
';
            };
};
object '2' : XmPushButton {
            arguments {
            lastGenName = 'button2';
            };
};
object '3' : XmPushButton {
            arguments {
            lastGenName = 'button3';
            };
};
object 'bruce_drawing' : XmDrawingArea {
            arguments {
            was_selected = true;
            lastGenName = 'bruce_drawing';
            generateResName = true;
            XmNforeground =  * color('SlateGrey');
            XmNbackground =  * color('GhostWhite');
            XmNmarginWidth =  * 400;
            XmNmarginHeight =  * 400;
            };
            callbacks {
            XmNinputCallback = '
bruce_drawing_callback_input( )
';
            XmNresizeCallback = '
bruce_drawing_callback_resize( )
';
            };
};
object '1' : XmMenuBar {
            arguments {
            lastGenName = 'menuBar1';
            };
object '1' : XmCascadeButton {
            arguments {
            lastGenName = 'cascade1';
            XmNlabelString = 'File';
            };
object '1' : XmPulldownMenu {
            arguments {
            lastGenName = 'menu1';
            };
```

Robot Visual Workshop (Motif) GUI-client: file: bp.xd. (cont.)

```
object '1' : XmPushButton {
        arguments {
        lastGenName = 'button1';
        XmNlabelString = 'Open';
        };
};
object '4' : XmPushButton {
        arguments {
        lastGenName = 'button4';
        XmNlabelString = 'Exit';
        };
};
};
};
object '2' : XmCascadeButton {
        arguments {
        lastGenName = 'cascade2';
        XmNlabelString = 'Edit';
        };
object '2' : XmPulldownMenu {
        arguments {
        lastGenName = 'menu2';
        };
};
};
};
};
};
end module;
```

```
/*********************************************************
 * File:  client.c

 Client.c is used to start the socket connection for the GUI-client.  It
    communicates with the server.  If communication is disrupted,
    it is re-started.  Data is sent and received.  Communication is hard coded to
    a predefined port number.  The server name is also hard-wired.
    This is just a prototype implementation.  A better method for
     doing for synchronizing the passing of data should be established.
 *
 * C/C++ version
 *
 *  Date:  8/97
 *
 * Compiler:  gcc or Sun Compiler
 ********************************************************* */

#include <stdio.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <errno.h>
#include <signal.h>
#include "global.h"

int inet();
void catchkill();
int int_client;
extern int errno;

FILE *fsock;
int gix,giy;
int gbNeedToStart = 0;

struct data sendmessage;
struct data recmessage;

/*********************************************
 Function: send_data
 Arguments: int x,     new x location for robot
            int y,     new y location for robot
 Return:  none
 Purpose: Used to request new x,y locations for robot.
*********************************************/
void send_data(int x, int y)
```

```
{
        gbNeedToStart = 1;
        gix = x;
        giy = y;
}

/*****************************************
 Function client_main
 Arguments: int *x, int *y;
 Return: none
 Purpose: Make connection to server, and it will return
        ix and iy if data is sent from the server.
*****************************************/
void client_main(int *ix, int *iy)
{
    int ii;
    int ret;

                ret = sock_data("pegasi.nosc.mil");
                if(ret < 0)
                        printf("could not write to socket\n");

                *ix = (float)recmessage.ix;
                *iy = (float)recmessage.iy;
}

/*****************************************
 Function sock_data
 Arguments: char traffic_host    (name of host to connect to)
 Return: int  ( -1 if fails )
 Purpose: Starts the socket connection,  functions
        called:(open, inet_client, inet)
*****************************************/
sock_data(traffic_host1)
void char *traffic_host1;
{
        static int ret1 = 0;
        static int sock1;  /*  inet socket file descriptor */
        static char unix_read[READ_SIZE];  /* array used to read message */
        static char old_parse1[READ_SIZE];  /* array to hold partial message */

                signal(SIGPIPE,catchkill);

                while(ret1 == 0)
                {
                        sock1 = inet_client(traffic_host1);
                        fsock = fdopen(sock1,"w");
                        if(sock1)
                                ret1 = 1;
                }
```

```
                if(ret1)
                        ret1 = inet(sock1); /*   inet socket */
                if(ret1 == 0)
                 return(-1);
                else
                 return(1);


/*****************************************
 Function inet_client
 Arguments: char traffic_host  (name of server to connect to
 Return:  int
 Purpose:  Continues setup of socket.
*****************************************/

inet_client(traffic_host)
        char *traffic_host;
{
        struct sockaddr_in myserver, from;
        int length = sizeof(from);
        struct hostent *hp;
        struct servent *sp;
        int pid;
        int ret;
        int  sock;

        signal(SIGPIPE,catchkill);

        if((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
                perror("could not make socket\n");
                exit(-1);
        }

        if((hp = gethostbyname(traffic_host)) == NULL) {
                perror(traffic_host);
                exit(-1);
        }
        myserver.sin_port = 4141;  /* Hard coded, should open services files */
        bcopy(hp->h_addr, &myserver.sin_addr.s_addr, hp->h_length);
        myserver.sin_family = AF_INET;

        if((ret = connect(sock, &myserver, sizeof(myserver))) >= 0) {
         printf("could  make connection \n");
         return(sock);
        }
         printf("could NOT make connection \n");
         sleep(1);
         close(sock);
         return(0);
```

```
}

/**********************************
 Function  inet
 Arguments: int sock (sock file descriptor)
 Return:  int
 Purpose: Reads/Writes on socket
**********************************************/
inet(sock)
   int sock;
{
        int ret;
      int bytes;
        char strg[100];

        printf("before Read\n");
        read(sock,&recmessage,sizeof(recmessage));
        printf("After Read\n");
      if(recmessage.ir == 1)
          {
           if(gbNeedToStart)
           {
            sendmessage.ir = 1;
            sendmessage.ix = gix;
            sendmessage.iy = giy;
            strg[0] = 1;
           }
        else
          {
           sendmessage.ir = 0;
           sendmessage.ix = 0;
           sendmessage.iy = 0;
           strg[0] = 65;
          }
        }
      else
          {
           sendmessage.ir = 0;
           sendmessage.ix = 0;
           sendmessage.iy = 0;
           strg[0] = 65;
          }
          if(sock > 0)
          if(ret=write(sock, strg, 1)<0)
          {
                  perror("error in write to sock");
                  close(sock);
                  return(0);   /* socket write failed return 0 */
          }
          if(sock > 0)
```

```
    if(recmessage.ir == 1)
        if(gbNeedToStart)
        {
         gbNeedToStart = 0;
         if(ret=write(sock, &sendmessage, sizeof(sendmessage))<0)
         {
                perror("error in write to sock");
                close(sock);
                return(0);   /* socket write failed return 0 */
         }
        }

        printf("After write\n");
        fflush(fsock);
        return(1);          /* socket write worked return 1 */
}

/*****************************************************
 Function:  catchkill
 Arguments: none
 Return:  void
 Purpose:. Catches unix signals.
*****************************************************/
void catchkill()
{
        printf("WE caught a signal\n");
}
```

Robot Visual Workshop (Motif) GUI-client: file: global.h

```
/*************************************************************
 * File:  global.h
 *
 * Contains the structure that is used for passing data
    on the socket for  the GUI-client.  This structure is modified
    for each application.
 *
 *
 * C/C++ version
 *
 * Date:  8/97
 ************************************************************* */
struct data
{
 int ix;
 int iy;
 int ir;
};
```

Robot Visual Workshop (Motif) GUI-client: file: lander.c

```
/*******************************************************
 * File:  lander.c
 *
 * lander.c does simple X-window drawing for the Visual Workshop (Motif)
 * GUI-client.
 *
 * C/C++ version
 *
 * Date:  8/97
 *******************************************************/

#include <X11/Xatom.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>

#include <Xm/Xm.h>
#include <Xm/DrawingA.h>
#include <Xm/PushB.h>

#include "bp.h"

Window gwindow;
Display *gdisplay = NULL;

/*******************************************************
 Function : drawline
 Arguments:  int x, int y, int x2, int y2
 Return:  void
 Purpose:. Draws a line from x,y to x2,y2  in the graphics window.
*******************************************************/
void drawline(int x,int y,int x2,int y2)
{
     GC          gc;
     Display     *display;
     Window      xid;
     int ii;

     xid = gwindow;
     display = gdisplay;
     gc = DefaultGC(display,DefaultScreen(display));

    XDrawLine(display,xid,gc,x,y,x2,y2);
}
```

```
/*****************************************************
 Function : drawcircle
 Arguments:  int x, y    location of circle
             int width
             int height
 Return:  void
 Purpose:.  Draw a circle a x,y with width and height.
*****************************************************/
void drawcircle(int x,int y,int width,int height)
{
     GC          gc;
     Display     *display;
     Window      xid;
     int ii;

     xid = gwindow;
     display = gdisplay;
     gc = DefaultGC(display,DefaultScreen(display));

          XDrawArc(display,xid,gc,x,y,width,height,0,64*360);

}

/*****************************************************
 Function : setup_image
 Arguments:  Wiget w
 Return:  void
 Purpose:. Gets display attributes from a widget.
*****************************************************/
void _image(Widget w)
{
 gdisplay  = XtDisplay(w);
 gwindow  = XtWindow(w);
}

/*****************************************************
 Function : do_image
 Arguments: int x,y
 Return:  int   (< 0 if error)
 Purpose:  Draw the robot at x,y
*****************************************************/
void do_image(int x,int y)
{
 static int oldx  = 0;
 static int oldy = 0;

 if(x == 0)
          return(0);
```

```
 if(gdisplay == NULL)
        return(-1);

 x = x/5 + 200;
 y = y/5 + 200;
 drawback();
 clearrect(oldx-11, oldy-11, 22, 22);
 drawlander(x,y);
 drawback();

 oldx = x;
 oldy = y;
}

/*******************************************************
 Function : clearrect
 Arguments: int x,y,width,height
 Return:  void
 Purpose: Clears the default display window at position x,y,width,
          height.
*******************************************************/
void clearrect(int x, int y, int w, int h)
{
 Window      xid;
 Display     *display;

     xid = gwindow;
     display = gdisplay;
 XClearArea(display,xid,x,y,w,h,0);
}


/*******************************************************
 Function : clear
 Arguments: none
 Return:  void
 Purpose:.
*******************************************************/
void clear()
{
 Window      xid;
 Display     *display;

     xid = gwindow;
     display = gdisplay;
     XClearWindow(display,xid);
}
```

```
/*****************************************************
 Function  draw_lander
 Arguments:  int x,y
 Return:  void
 Purpose:. Draw the actual lander.
*****************************************************/
void drawlander(int x, int y)
{
      drawcircle(x-5,y-5,10,10);
      drawline(x,y-8,x,y+8);
      drawline(x-8,y,x+8,y);
}

/*****************************************************
 Function : drawback
 Arguments: none
 Return:  void
 Purpose: Draw the screen background.
*****************************************************/
void drawback()
{
          drawcircle(0,0,400,400);
          drawcircle(196,196,8,8);
          drawcircle(198,198,4,4);

}
```

Robot Visual Workshop (Motif) GUI-client: file: th1.c

```
/*********************************************************
 * File:  th1.c
 *
 * th1.c is used to create a thread of execution to read
 * on the socket, and register repaint events, for the GUI-client.
 *
 * C/C++ version
 *
 * Date:  8/97
 ********************************************************* */

#include <stdio.h>
#include <math.h>
#include <thread.h>
#include <synch.h>
#include <errno.h>

        thread_t twriter;

        int x = 0;
        int y = 0;

/*********************************************************
 Function:  add
 Arguments:  none
 Return:  none
 Purpose: Add is called by the second.  Add never returns.  Add is
          in an infinite loop that calls the socket function (client_main)
          to get data from the PSDL-server.
          The image is then drawn.
*********************************************************/
void add()
{
        while(1)
        {
                client_main(&x,&y);
                do_image(x,y);
        }
}

void sub()
{
                fprintf(stderr,"Sub\n");
}
```

Robot Visual Workshop (Motif) GUI-client: file: th1.c (cont.)

```
/*****************************************************
 Function : oldmain
 Arguments: none
 Return: void
 Purpose: Starts the second thread.  Add will be called by the
          second thread.
*****************************************************/
void oldmain()
{
        int     i;

        thr_setconcurrency(2);
        thr_create(NULL,NULL,(VPTR)add,NULL,THR_NEW_LWP,&twriter);
}
```

LIST OF REFERENCES

Bruce D. Plutchak, plutchak@nosc.mil

THE DESIGN OF AN INTERFACE EDITOR FOR THE COMPUTER-AIDED PROTOTYPING SYSTEM

1. Luqi, Shing, M., "CAPS: A Tool for Real-Time System Development and Acquisition", *National Research Review,* 1992

2. Luqi, Ketabchi, M., "A Computer-Aided Prototyping System", *IEEE Software,* March 1988

3. Luqi, Berzins, V., "A Prototyping Language for Real-Time Software", *IEEE Transactions on Software Engineering,* October 1988

4. Luqi, Berzins, V., "Rapidly Prototyping Real-Time Systems", *IEEE Software,* March 1988

5. Luqi, Steigerwadl, R., Hughes, G., Berzins, V., "CAPS as a Requirements Engineering Tool", *Proceedings of Tri-Ada Conference,* October 1991

6. Luqi, "Software Evolution Through Rapid Prototyping", *IEEE Computer,* May 1995

7. Luqi, Goguen, J., "Formal Methods: Promises and Problems", *IEEE Software,* January 1997

8. Pace, "CAPS Release 1 Tutorial", Thesis, Naval Postgraduate School, Monterey, California

9. Dixon, R., "The Design and Implementation of a User Interface for the Computer-Aided Prototyping System", Thesis, Naval Postgraduate School, Monterey, California, September 1992

10. *TAE: Getting Started with TAE Plus*, *TAE Plus Reference Manual V 5.2*, NASA, December 1992

11. Myers., B., "UIMSs, Toolkits, Interface Builders", *ACM Transactions on Computer-Human Interaction,* March 1995

12. Valaer, L., Babb II, R., "Choosing a User Interface Development Tool", *IEEE Software,* July 1997

13. Smart, J., "wxWindows", wxWindows *Home Page*, http://web.ukonline.co.uk/julian.smart/wxwindows.com

14. Armstrong Jr., J., "GUI Tools Mature", *Advanced Systems,* March 1995

15. *OpenUI Overview,* Open Software Associates, January 1996

16. Aonix *Home Page,* http://www.aonix.com

17. Visual Numerics *Home Page,* http//www.vni.com

18. MathWorks Inc. *Home Page,* http:/www.mathworks.com

19. DataViews Corp. *Home Page,* http://www.dvcorp.com

20. "Galaxy Application Environment", Visix Software Inc., 1995

21. Sun Microsystems *Home Page,* http://www.sun.com

22. McKay, R., "Platform Independent FAQ", zeta.org.au *Home Page*, http://www.zeta.org.au/~rosko/pigui.htm, 1997

23. XVT Software *Home Page,* http://www.xvt.com

24. "Three Tier Software", Tier3 *Home Page*, http://www.tier3.com/Make3/threet.htm

25. Myers, B., "User Interface Software Tools", CMU *Home Page,* http://www.cs.cmu.edu/afs/cs.cmu.edu/user/bam/www/toolnames.html

26. Elliott, R., Powers, N., "One-Tier, Two-Tier, Three-Tier, A Server: Using Technology to Solve Business Problems", Pacific-Electric *Home Page*, http://www.pacific-electric.com/PacificElec/Product/whtpap04.htm

27. Churchill, S., "Programming Fresco, A Hands-On Tutorial to the Fresco User Interface System", Faslab *Home Page*, http://www.faslab.com

28. Open Group *Home Page,* http:/www.opengroup.org/tech.desktop/x

29. JavaSoft *Home Page,* http://www.javasoft.com

30. Heller, P., Roberts, S., *Java 1.1 Developer's Handbook*, Sybex, 1997

31. Imperial Software Technology *Home Page,* http://www.ist.co.uk

32. Orfali, R., Harkey, D., *Client/Server Programming with Java and CORBA,* Wiley & Sons, 1997

33. Vogel, A., "Socket Programming with Java", *Java Developer's Journal,* Vol. 2 Issue: 4 1997

34. *Symantec Visual Café - Getting Started,* Symantec Corporation, 1996

35. Ruiz, N., "Smartship Project Modeling", Thesis, Naval Postgraduate School, Monterey, California

36. Cohn, M., et al., *Java Developer's Reference,* Sams Publishing, 1996

37. Sutherland, J., "The Java Revolution", *SunExpert,* January 1997

38. Visual Edge *Home Page,* http://www.vedge.com

39. Integrated Computer Solutions *Home Page,* http://www.ics.com

40. Open Software Associates *Home Page,* http://www.osa.com

41. *Visual Workshop User's Guide Version 2,* SunSoft, 1996

42. *Xda: Ada 95 for X-Designer,* OC-Systems *Home Page*, http://ocsystems.com

43. Parker, T., "Visual Workshop 3.0 for C++", *UNIX Review*, July 1997

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center..... 2
   8725 John J. Kingman Rd., Suite 0944
   Fort Belvoir, VA 22060-6218

2. Dudley Knox Library.....2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Center for Naval Analysis..... 1
   4401 Ford Ave.
   Alexandria, VA 22302

4. Dr. Ted Lewis, Chairman, Code CS/Lt..... 1
   Computer Science Dept.
   Naval Postgraduate School
   833 Dyer Rd.
   Monterey, CA 93943

5. Chief of Naval Research..... 1
   800 North Quincy St.
   Arlington, VA 22217

6. Dr. Luqi, Code CS/Lq.....1
   Computer Science Dept.
   Naval Postgraduate School
   833 Dyer Rd.
   Monterey, CA 93943

7. Dr. V. Berzins, Code CS/Be..... 1
   Computer Science Dept.
   Naval Postgraduate School
   833 Dyer Rd.
   Monterey, CA 93943

8. Dr. Marvin Langston..... 1
   1225 Jefferson Davis Highway
   Crystal Gateway 2 / Suite 1500
   Arlington, VA 22202-4311

9. David Hislop..... 1
Army Research Office
PO Box 12211
Research Triangle Park, NC 27709-2211

10. Capt. Talbot Manvel..... 1
Naval Sea System Command
2531 Jefferson Davis Hwy.
Attn: TMS 378 Capt. Manvel
Arlington, VA 22240-5150

11. CDR Michael McMahon..... 1
Naval Sea System Command
2531 Jefferson Davis Hwy.
Arlington, VA 22242-5160

12. Dr. Elizabeth Wald..... 1
Office of Naval Research
800 N. Quincy St.
ONR CODE 311
Arlington, VA 22217-5660

13. Dr. Ralph Wachter..... 1
Office of Naval Research
800 N. Quincy St.
CODE 311
Arlington, VA 22217-5660

14. Army Research Lab..... 1
115 O'Keefe Building
Attn: Mark Kendall
Atlanta, GA 30332-0862

15. National Science Foundation..... 1
Attn: Bruce Barnes
Div. Computer & Computation Research
1800 G St. NW
Washington, DC 20550

16. National Science Foundation..... 1
Attn: Bill Agresty
4201 Wilson Blvd.
Arlington, VA 22230

17. Hon. John W. Douglass..... 1
Assistant Secretary of the Navy
(Research, Development and Acquisition)
Room E741
1000 Navy Pentagon
Washington, DC 20350-1000

18. Technical Library Branch..... 1
Naval Command, Control, and Ocean Surveillance Center
RDT&E Division, Code D0274
San Diego, CA 92152-5001

19. Naval Command, Control and Ocean Surveillance Center..... 1
Attn: Steve Nunn
RDT&E Division, Code D44213
53245 Patterson Rd.
San Diego, CA 92152-7150

20. Naval Command, Control and Ocean Surveillance Center..... 1
Attn: Bruce Plutchak
RDT&E Division, Code D44213
53245 Patterson Rd.
San Diego, CA 92152-7150